TOWARDS VERIFYING NEURAL NETWORKS AGAINST MULTI-PARAMETER BIT-FLIP PERTURBATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hardware faults can flip bits in the stored weights of a quantized neural network, potentially compromising its predictions. While such faults typically affect multiple parameters simultaneously, existing verifiers are limited to single-parameter perturbations due to the combinatorial explosion of possible flip locations in large networks. We present mBFV (m-BitFlip Verifier), an efficient verification framework that proves robustness against simultaneous bit flips across multiple parameters without explicitly enumerating these combinations. mBFV achieves this via a novel multi-bitflip bound propagation technique that directly aggregates the m worst-case contributions. To further tighten these bounds, mBFV employs a branch-and-bound mechanism over perturbation locations, partitioning the potential flips to smaller groups of neurons. Evaluated on 625 instances, mBFV successfully verifies 293, significantly outperforming a prior single-parameter verifier (38 verified instances) and an exact mixed-integer linear programming baseline (0 verified instances). Notably, while these baselines are restricted to single-parameter flips on small networks (up to 13k parameters), mBFV scales to verify networks with up to 1.15M parameters against up to four simultaneous parameter flips.

1 INTRODUCTION

Neural networks are widely deployed with quantized weights across diverse hardware platforms, ranging from edge microcontrollers (Jacob et al., 2018) to data center accelerators (Frantar et al., 2023). However, the physical memory storing these discrete values is vulnerable to hardware level bit flips, whether induced maliciously via DRAM Rowhammer (Kim et al., 2014) and fault injection (Breier et al., 2018), or naturally through radiation soft errors (Schmedding et al., 2024). Consequently, a few flipped bits in the weights can cause a network to misclassify (Hong et al., 2019; Rakin et al., 2019). Empirical testing is necessary but insufficient to show that such flips do not exist because the finite set of flip combinations grows exponentially with the number of flipped parameters. In contrast, neural network verification (NNV) can prove that a property holds for every possible perturbation (Katz et al., 2017; Liu et al., 2021; Singh et al., 2019; Zhang et al., 2018).

BFAVerifier (Zhang et al., 2025) is currently the state-of-the-art (SOTA) verifier for bit flips, yet it is strictly limited to a single perturbed parameter. However, a single flipped parameter rarely changes the prediction of a quantized network (Hong et al., 2019). Known single-flip vulnerabilities need stronger assumptions such as control of training (Dong et al., 2023) or a planted input trigger (Li et al., 2025). In practice, flips that change the prediction come in chains of 11 to 17 flips (Rakin et al., 2019) or 2 to 24 flips (Yao et al., 2020). Therefore, a verification certificate for a single parameter can give false assurance. In fact, in our evaluation (§5.2) , only 74 of the 121 instances that mBFV verifies with one flipped parameter remain verified with four.

Verifying multiple flipped parameters is inherently more challenging because there are $\binom{K}{m}$ ways to choose m of K parameters (§4.2). For $m = 2$, a network with 5,000 parameters already has about 12.5 million pairs of parameters. Running BFAVerifier (Zhang et al., 2025) once per pair is infeasible even for such a small network. A *complete verifier* decides every instance given enough time, whereas an incomplete verifier certifies only some robust instances and returns unknown for the rest (Tjeng et al., 2017). An MILP encoding (Tjeng et al., 2017) is complete and covers all choices at once, but it does not scale beyond small networks (§5). To our knowledge, no existing verifier handles several flipped parameters on networks with millions of parameters.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this paper, we introduce mBFV (m BitFlip Verifier), an efficient incomplete verifier that employs both bounding and branching to formally verify quantized networks against multiple parameter bit flip perturbations. Our key observation is that bound propagation, which is the basis of verifiers such as CROWN (Zhang et al., 2018) and DeepPoly (Singh et al., 2019), can approximate the network output variation as a sum containing one distinct contribution per perturbed parameter. When at most m parameters may flip, the worst case output deviation is simply the aggregate of the m largest individual contributions. Therefore, our *multiple-parameter bound propagation* algorithm computes this maximal sum directly without enumerating every possible parameter combination. A classical result from combinatorial optimization (Schrijver, 2003) guarantees that this direct aggregation method incurs no loss of precision relative to the isolated parameter contributions.

To improve precision, mBFV employs a branch and bound (BaB) strategy over the perturbed parameters. Because each parameter contributes to exactly one neuron, altering m parameters affects a maximum of m neurons. mBFV leverages this structural property by isolating potential flips to small groups of neurons while keeping the remaining neurons to their original values, which allows multiple-parameter bound propagation to compute significantly tighter bounds. Note that the generated subproblems cover the entire search space to ensure that no combination of flipped parameters is omitted. The algorithm iteratively partitions any subproblem that bound propagation fails to verify until the entire space is proven safe. More importantly, for a network containing ν neurons distributed across L layers, this branching strategy generates a maximum of $1 + \binom{2\nu-L}{m}$ subproblems. Since each subproblem requires only a single bound propagation, the overall verification complexity scales polynomially with the number of neurons (Prop. 4.1).

We evaluate mBFV againts $m \in \{1, 2, 4\}$ flipped parameters on fully-connected networks for ACAS Xu, MNIST, and CIFAR-10 with up to 1.15M parameters (Brix et al., 2023). Across 625 total instances, mBFV with BaB proves 293 instances against 184 for $\text{mBFV}_{\text{w/o BaB}}$, 38 for BFAVerifier, and none for the exact MILP. Notably, while the SOTA baseline is restricted to single parameter perturbations on small networks containing at most 13k parameters, mBFV scales effectively to verify architectures with up to 1.15M parameters and up to four simultaneous parameter flips.

Our contributions include: (i) multiple parameter bound propagation, the core bounding mechanism of mBFV, which evaluates m flipped parameters simultaneously by aggregating their m largest individual contributions (§3), (ii) a branch and bound strategy over perturbation locations that restricts potential flips to small groups of neurons, ensuring the total number of bounded subproblems scales polynomially with the number of neurons (§4), and (iii) an GPU implementation and an evaluation on architectures containing up to 1.15 million parameters (§5).

2 Background

Quantized neural networks. A quantized neural network (QNN) with L layers and K parameters is a function $N : \mathbb{R}^d \to \mathbb{R}^s$. Each parameter w of layer i is stored as the signed Q-bit integer

$$q(w) = \text{clip}\big(\text{round}(w/\Delta_i),\ -(2^{Q-1}-1),\ 2^{Q-1}-1\big) \tag{1}$$

where the step size Δ_i is the largest absolute value of a real parameter of layer i divided by $2^{Q-1}-1$, round rounds to the nearest integer, and the clip applies only at quantization time. The integer is stored in two's complement $\vec{v}(q) = (v_Q, \ldots, v_1) \in \{0,1\}^Q$ with sign bit v_Q and $q = -2^{Q-1}v_Q + \sum_{t<Q} 2^{t-1}v_t$, and it is de-quantized to $\Delta_i\, q(w)$ at inference. With the de-quantized weights $W^{(i)}$ and biases $b^{(i)}$, the network computes $z_0 = x$, $\hat{z}_i = W^{(i)}z_{i-1} + b^{(i)}$ (pre-activation), $z_i = \sigma(\hat{z}_i)$ (post-activation) with an activation σ such as ReLU for $i < L$, and $N(x) = \hat{z}_L$, where layer i has ν_i neurons.

Definition 2.1 (Perturbation vector)**.** Given integers m and $n \le Q$, an (m, n)*-perturbation vector* $\rho = \{(\alpha_t, P_t) \mid t \le m\}$ is a set of at most m pairs with pairwise distinct parameters α_t (weights or biases of N). Each $P_t \subseteq \{1, \ldots, Q\}$ is a possibly empty set of at most n bit positions of $\vec{v}(q(\alpha_t))$. The network N^ρ is obtained from N by flipping these bits, and $\mathcal{P}_{m,n}$ denotes the set of all (m, n)-perturbation vectors.

Under at most n bit flips, a parameter α of layer i can take exactly the values

$$R_n(\alpha) = \big\{\, \Delta_i \,\text{dec}\big(\vec{v}(q(\alpha)) \oplus e\big) \ :\ e \in \{0,1\}^Q,\ |e| \le n \,\big\} \tag{2}$$

Alg. 1: Bound propagation under up to m flipped parameters

Input: network N (weights W, biases b), property $\phi_{in} \Rightarrow \phi_{out}$, parameter intervals $[\ell_p, u_p]$, numbers of perturbed parameters m

Output: $lb \leq f^\star$

1 $[\underline{z}_0, \overline{z}_0] \leftarrow \phi_{in}$
2 **for** $\ell = 1, \ldots, L$ **do**
3 **foreach** *neuron* t *of layer* ℓ **do**
4 $\underline{G}, \overline{G} \leftarrow \emptyset$
5 **for** $i = \ell, \ldots, 1$ **do**
6 **foreach** *parameter* $p = (j, k)$ *of layer* i **do**
7 $\underline{G} \leftarrow \underline{G} \cup \{\min\{\underline{A}_j \, D \, z : D \in \{\ell_p, u_p\},\ z \in \{\underline{z}_{i-1,k}, \overline{z}_{i-1,k}\}\}\}$ // Eq. 9
8 $\overline{G} \leftarrow \overline{G} \cup \{\max\{\overline{A}_j \, D \, z : D \in \{\ell_p, u_p\},\ z \in \{\underline{z}_{i-1,k}, \overline{z}_{i-1,k}\}\}\}$
9 $(\underline{A}, \underline{b}, \overline{A}, \overline{b}) \leftarrow \textsc{Relax}\big(\underline{A}, \underline{b}, \overline{A}, \overline{b}, [\hat{\underline{z}}_{i-1}, \hat{\overline{z}}_{i-1}]\big)$
10 $\hat{\underline{z}}_{\ell,t} \leftarrow \min_{x \in \phi_{in}}(\underline{A}x + \underline{b}) + \text{top}_m(\underline{G})$ // Prop. 3.1
11 $\hat{\overline{z}}_{\ell,t} \leftarrow \max_{x \in \phi_{in}}(\overline{A}x + \overline{b}) - \text{top}_m(-\overline{G})$
12 $[\underline{z}_\ell, \overline{z}_\ell] \leftarrow \textsc{Act}\big([\hat{\underline{z}}_\ell, \hat{\overline{z}}_\ell]\big)$
13 **return** $\min_t \hat{\underline{z}}_{L,t}$

where $\oplus$ is bitwise XOR, $|e|$ counts the flipped bits, and dec decodes two's complement. This set contains the *nominal* value $\Delta_i \, q(\alpha)$ without flips ($e = 0$) and sign-bit flips, which can reach $-2^{Q-1}\Delta_i$ beyond the clip range of Eq. 1. We enumerate the set exactly and use its interval hull, the *change interval*

$$[\ell_\alpha, u_\alpha] = \big[\min R_n(\alpha) - \Delta_i \, q(\alpha),\ \max R_n(\alpha) - \Delta_i \, q(\alpha)\big], \qquad \ell_\alpha \leq 0 \leq u_\alpha \tag{3}$$

DNN Verification. NNV techniques prove that every input satisfying a property ϕ_{in} yields an output satisfying a property ϕ_{out}. Standard NNV problems take ϕ_{in} as a box $\mathcal{B} = \{x : \|x - x_0\|_\infty \leq \epsilon\}$ and ϕ_{out} as $N(x)_g > N(x)_k$ for every $k \neq g$, where g is the label of x_0. The property thus holds if and only if the smallest margin $f(x) = \min_{k \neq g}\big(N(x)_g - N(x)_k\big)$ stays positive on the box

$$\min_{x \in \mathcal{B}} \ f(x) \ > \ 0 \tag{4}$$

Under bit flips, we require Eq. 4 for every perturbed network N^ρ with $\rho \in \mathcal{P}_{m,n}$:

$$f^\star \ := \ \min_{\rho \in \mathcal{P}_{m,n}} \ \min_{x \in \mathcal{B}} \ f^\rho(x) \ > \ 0 \tag{5}$$

A verifier is *sound* if it never verifies a problem with $f^\star \leq 0$. A *complete* verifier also decides every problem, *e.g.*, by solving (5) exactly as an MILP (§C.1). An *incomplete* verifier such as mBFV computes a lower bound $lb \leq f^\star$. If $lb > 0$, it returns *verified*, since no perturbation $\rho \in \mathcal{P}_{m,n}$ and input $x \in \mathcal{B}$ violate the property. Otherwise it returns *unknown*.

3 Multi-Parameter Bound Propagation

mBFV extends CROWN (Zhang et al., 2018), which bounds each neuron as $\underline{A}x + \underline{b} \leq \hat{z}_j \leq \overline{A}x + \overline{b}$, so that these bounds also hold when up to m parameters flip. We call this extension *multi-bitflip bound propagation* (Alg. 1). At each layer ℓ (line 2), mBFV bounds every pre-activation neuron (line 3) by *back-substitution* to the input (line 5). During the back-substitution, it records the smallest and the largest term (Eq. 9) of every parameter in $\underline{G}$ and $\overline{G}$ (line 7-line 8, §3.1) and relaxes the activation (line 9). Next, it *concretizes* both bounds over ϕ_{in} (§3.2) and adds the sum of the m smallest terms to the lower bound (line 10) and the sum of the m largest terms to the upper bound (line 11). Activation Act then maps the resulting pre-activation ranges for the next layer (line 12). Finally, mBFV returns the lower bound of ϕ_{out} (line 13).

3.1 PER-PARAMETER DECOMPOSITION

Back-substitution bounds an output $\hat{z}_{\ell,t}$ in terms of each earlier layer i:

$$\underline{A}\,\hat{z}_i + \underline{b} \;\le\; \hat{z}_{\ell,t} \;\le\; \overline{A}\,\hat{z}_i + \overline{b} \tag{6}$$

The process starts at $i = \ell$ and ends at the input x, where the bounds become linear functions of x, *e.g.*, $\underline{A}x + \underline{b} \;\le\; \hat{z}_{\ell,t} \;\le\; \overline{A}x + \overline{b}$. Each step from layer i to layer $i-1$ replaces $\hat{z}_i$ in Eq. 6 by its definition, which under bit flips is

$$\hat{z}_i = (W^{(i)} + D^{(i)})\, z_{i-1} + b^{(i)} + d^{(i)} \tag{7}$$

where each change $D^{(i)}_{jk}$ or $d^{(i)}_j$ is 0 if its parameter p is not perturbed and otherwise lies in its change interval $[\ell_p, u_p]$ Eq. 3. Substituting Eq. 7 into Eq. 6 gives

$$\begin{aligned}
\underline{A}\,\hat{z}_i + \underline{b} &= \underbrace{\underline{A}\big(W^{(i)} z_{i-1} + b^{(i)}\big) + \underline{b}}_{\text{nominal}} + \sum_{j,k} \underline{A}_j D^{(i)}_{jk} z_{i-1,k} + \sum_j \underline{A}_j d^{(i)}_j \\
\overline{A}\,\hat{z}_i + \overline{b} &= \overline{A}\big(W^{(i)} z_{i-1} + b^{(i)}\big) + \overline{b} + \sum_{j,k} \overline{A}_j D^{(i)}_{jk} z_{i-1,k} + \sum_j \overline{A}_j d^{(i)}_j
\end{aligned} \tag{8}$$

The nominal part is back-substituted as in CROWN (Zhang et al., 2018), while the remaining terms carry the flips. Since the flip of each weight $p = W^{(i)}_{jk}$ is unknown, mBFV replaces its terms $\underline{A}_j D^{(i)}_{jk} z_{i-1,k}$ and $\overline{A}_j D^{(i)}_{jk} z_{i-1,k}$ by their worst cases

$$\begin{aligned}
\underline{g}_p &= \min_{D^{(i)}_{jk} \in [\ell_p, u_p],\; z_{i-1,k} \in [\underline{z}_{i-1,k}, \overline{z}_{i-1,k}]} \underline{A}_j\, D^{(i)}_{jk}\, z_{i-1,k} \\
\overline{g}_p &= \max_{D^{(i)}_{jk} \in [\ell_p, u_p],\; z_{i-1,k} \in [\underline{z}_{i-1,k}, \overline{z}_{i-1,k}]} \overline{A}_j\, D^{(i)}_{jk}\, z_{i-1,k}
\end{aligned} \tag{9}$$

Both extremes lie at corners of $[\ell_p, u_p] \times [\underline{z}_{i-1,k}, \overline{z}_{i-1,k}]$ because the term is bilinear (line 7). Likewise, the terms $\underline{A}_j d^{(i)}_j$ and $\overline{A}_j d^{(i)}_j$ of each bias $p = b^{(i)}_j$ have the worst cases

$$\underline{g}_p = \min_{d^{(i)}_j \in [\ell_p, u_p]} \underline{A}_j\, d^{(i)}_j \qquad \overline{g}_p = \max_{d^{(i)}_j \in [\ell_p, u_p]} \overline{A}_j\, d^{(i)}_j \tag{10}$$

Let the vector $s \in \{0,1\}^K$ indicate the perturbed parameters, with $s_p = 1$ if parameter p is perturbed and $s_p = 0$ otherwise. For every input $x \in \mathcal{B}$ and every choice of perturbed parameters, mBFV bounds $\hat{z}_{\ell,t}$ by

$$\underline{A}x + \underline{b} + \sum_p s_p\, \underline{g}_p \;\le\; \hat{z}_{\ell,t} \;\le\; \overline{A}x + \overline{b} + \sum_p s_p\, \overline{g}_p \tag{11}$$

3.2 MULTI-PARAMETER CONCRETIZATION

Concretization turns Eq. 11 into scalars by optimizing over every input $x \in \mathcal{B}$ and every choice of at most m perturbed parameters $\mathcal{S}_m = \Big\{ s \in \{0,1\}^K : \sum_p s_p \le m \Big\}$. The input part and the perturbation part of Eq. 11 share no variables, so the bounds split into:

$$\begin{aligned}
\underline{\hat{z}}_{\ell,t} &= \underbrace{\min_{x \in \mathcal{B}} \big(\underline{A}x + \underline{b}\big)}_{\text{input part}} + \underbrace{\min_{s \in \mathcal{S}_m} \sum_p s_p\, \underline{g}_p}_{\text{perturbation part}} \\
\overline{\hat{z}}_{\ell,t} &= \max_{x \in \mathcal{B}} \big(\overline{A}x + \overline{b}\big) + \max_{s \in \mathcal{S}_m} \sum_p s_p\, \overline{g}_p
\end{aligned} \tag{12}$$

The input part is concretized as in CROWN (Zhang et al., 2018). The perturbation part is an integer program over $\binom{K}{m}$ choices of parameters, which mBFV relaxes to the linear program over the convex hull of $\mathcal{S}_m$:

$$\mathcal{C}_m = \Big\{ s \in [0,1]^K : \sum_p s_p \le m \Big\} \tag{13}$$

Prop. 3.1 shows that the relaxation is exact, so linear and integer programs have the same optimum:

$$\begin{aligned} \min_{s\in\mathcal{C}_m} \sum_p s_p\,\underline{g}_p &= \min_{s\in\mathcal{S}_m} \sum_p s_p\,\underline{g}_p = \operatorname{top}_m(\underline{g}) \\ \max_{s\in\mathcal{C}_m} \sum_p s_p\,\overline{g}_p &= \max_{s\in\mathcal{S}_m} \sum_p s_p\,\overline{g}_p = -\operatorname{top}_m(-\overline{g}) \end{aligned} \tag{14}$$

where $\operatorname{top}_m(g)$ is the sum of the m smallest entries of g, and the upper bound follows by applying Prop. 3.1 to $-\overline{g}$.

Proposition 3.1. *Assume $g_p \le 0$ for every p. Let T be a set of $\min(m, K)$ parameters whose terms are no larger than any term outside T (the m most negative terms). Then*

(a) $\sum_{p\in T} g_p \le \sum_p s_p g_p$ for every $s \in \mathcal{C}_m$;
(b) the 0/1 vector $\mathbf{1}_T$ of T lies in $\mathcal{S}_m$ and attains $\sum_{p\in T} g_p$.

Hence $\min_{s\in\mathcal{C}_m} \sum_p s_p g_p = \min_{s\in\mathcal{S}_m} \sum_p s_p g_p = \sum_{p\in T} g_p = \operatorname{top}_m(g)$.

Proof sketch. Pick a threshold $\theta \le 0$ with $g_p \le \theta$ inside T and $g_p \ge \theta$ outside. Then $(s_p - \mathbf{1}_T(p))(g_p - \theta) \ge 0$ elementwise, and summing gives $\sum_p s_p g_p - \sum_{p\in T} g_p \ge \theta(\sum_p s_p - |T|) \ge 0$. The vector $\mathbf{1}_T$ attains the bound. §B gives the full proof, checked in Lean. □

For example, with $g = (-5, -3, -1, 0)$ and $m = 2$, the optimum is $s = (1, 1, 0, 0)$ with value -8, while $s = (1, 0.5, 0.5, 0)$ reaches only -7. The perturbation part thus has a closed form, whereas word substitutions need dynamic programming (Xu et al., 2020).

3.3 Soundness of mBFV

We show that every bound of Alg. 1 holds for every input $x \in \mathcal{B}$ and every flip in at most m parameters. Assume this holds for the neuron ranges of the earlier layers, which Alg. 1 computes first, so CROWN's relaxation of the activation is valid. Each flipped parameter then contributes a term between $\underline{g}_p$ and $\overline{g}_p$ to Eq. 8, since its change lies in $[\ell_p, u_p]$ and its input lies in its range. Every other parameter contributes 0, so Eq. 11 holds with the vector $s \in \mathcal{S}_m$ of the flipped parameters. Because Eq. 12 optimizes over all inputs and all $s \in \mathcal{S}_m$, the true value of $\hat{z}_{\ell,t}$ lies between $\underline{\hat{z}}_{\ell,t}$ and $\overline{\hat{z}}_{\ell,t}$. By induction over the layers, mBFV is therefore sound.

4 Branch and Bound for Bit Flip Verification

Branch and bound tightens the bounds of multi-bitflip bound propagation (§3) by restricting where the flips may happen. Each *subproblem* allows flips only in a few chosen neurons, so every other neuron keeps its original range (§4.1). Fewer changing neurons give tighter ranges in the following layers and hence a tighter bound on the margin. The network is verified when every subproblem is verified, and each subproblem that is not verified is split into smaller ones.

Alg. 2 describes the search, where BOUND is the lower bound on the margin that multi-bitflip bound propagation computes when flips are allowed only in the given neurons. It first allows flips in all neurons (line 1) and otherwise bounds the subproblems in a queue (line 3 to line 6). It splits every subproblem whose bound is not positive (line 10) and returns verified once all subproblems are verified (line 12).

4.1 Groups and Subproblems

The parameters of a neuron $\hat{z}_j$ are its incoming weights and bias, and they feed only $\hat{z}_j$ (Fig. 1a). Since the linear relaxation allow flips to occur anywhere, multi-bitflip bound propagation therefore widens the range of every neuron (Fig. 1b). Consider first m flips in a single layer. A *group* is a set of neurons of one layer, and allowing flips only in a group widens only its neurons (Fig. 1c). All other neurons keep their original ranges, which tightens the ranges in the following layers. Flips spread over several layers can be solved by solving multiple a single layer multi-bitflip.

Alg. 2: Branch and bound over perturbation locations

Input: network N, property $\phi_{in} \Rightarrow \phi_{out}$, number m of flipped parameters, time limit
Output: verified, unknown, or timeout

1 **if** BOUND(*all neurons*) > 0 **then** // §3
2 **return** *verified*
3 $Q \leftarrow \{S \subseteq \{\text{layers}\} : |S| = m\}, \quad \mathcal{G} \leftarrow Q, \quad P \leftarrow \emptyset$ // §4.1
4 **while** $Q \neq \emptyset$ **do**
5 pop $S \in Q$ with $\bigcup S \not\subseteq \bigcup S'$ for all $S' \in P$ // §4.2
6 **if** BOUND$(S) > 0$ **then** // §3
7 $P \leftarrow P \cup \{S\}$
8 **else**
9 **if** $|G| = 1$ *for all* $G \in S$ **then return** *unknown*
10 $Q \leftarrow Q \cup (\text{BRANCH}(S) \setminus \mathcal{G}), \quad \mathcal{G} \leftarrow \mathcal{G} \cup \text{BRANCH}(S)$ // Eq. 15
11 **if** *time limit reached* **then return** *timeout*
12 **return** *verified*

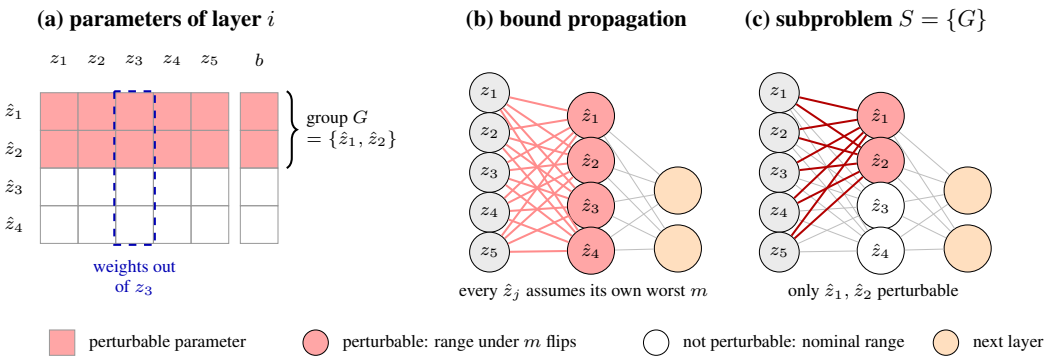

Fig. 1: Allowing flips only in a group of neurons of layer i.

A *subproblem* S is a set of at most m disjoint groups, and a perturbation *lies in* S if it flips only parameters of neurons in these groups. BOUND(S) allows flips only in these neurons, so it bounds the margin under every perturbation that lies in S. The search starts with whole layers as groups (line 3), since m flipped parameters lie in at most m layers.

When S is not verified, BRANCH(S) splits its largest group G at the middle neuron into two parts G_1 and G_2 and returns the subproblems

$$\text{BRANCH}(S) = \{C \subseteq T : |C| = m\}, \qquad T = (S \setminus \{G\}) \cup \{G_1, G_2\} \tag{15}$$

Each of the m flipped parameters lies in exactly one group of T, so every perturbation that lies in S also lies in one of these subproblems (Prop. A.1). If T has at most m groups, the only such subproblem allows the same neurons as S, so BRANCH splits again. Unlike branching on activation phases (Wang et al., 2021) or on the input box, this search branches on the remaining multi-bitflip relaxed neurons.

4.2 SOUNDNESS AND COMPLEXITY

Soundness. At every step of Alg. 2, each perturbation in $\mathcal{P}_{m,n}$ lies in a subproblem of the queue Q or of the verified set P, since BRANCH keeps every perturbation (Prop. A.2). A subproblem $S \in P$ has BOUND$(S) > 0$, so the margin stays positive under every perturbation that lies in S. Let S' be a subproblem in Q whose neurons are all neurons of some $S \in P$. Every perturbation that lies in S' then also lies in S and is already verified, so the search skips S' without bounding it. When Q is empty, every perturbation therefore lies in a subproblem of P, so the algorithm is sound (Prop. A.3).

Complexity. The cost of Alg. 2 is the number of calls of BOUND, each of which runs multi-bitflip bound propagation once. Prop. 4.1 bounds this number in terms of the number of neurons ν (§3.3).

Proposition 4.1 (Number of bound propagations)**.** *For every m, batch size, time limit, and outcome of* BOUND*, and whatever the verdict,* mBFV *makes at most*

$$1 + \binom{2\nu - L}{m} \tag{16}$$

bound propagations. Assume $\nu > m$, no time limit, and BOUND$(S) > 0$ *exactly when S may perturb the parameters of at most m neurons. Then* mBFV *returns verified after at least $1 + \binom{\nu}{m}$ bound propagations, and for $m = 1$ and $L \geq 2$ after exactly $1 + 2\nu - L$, so Eq. 16 is attained.*

The count is polynomial in ν, whereas there are $\binom{K}{m}$ choices of m parameters. For MNIST FC 2×256 ($\nu = 522$, $L = 3$) and $m = 2$, the bound is $541{,}321$ bound propagations, against $3.6 \cdot 10^{10}$ pairs of parameters. §A.3 gives the proof, whose counting is checked in Lean.

5 EVALUATION

5.1 BENCHMARKS AND SETUP

Benchmarks and Fine-tuning. We evaluate mBFV on seven architectures (Brix et al., 2023; Zhang et al., 2025): three ACAS Xu networks (6×50), three fully connected MNIST networks (2×256, 4×256, 7×250), and a CIFAR-10 network (7×250, 1.15M parameters).

Following BFAVerifier (Zhang et al., 2025), all networks are quantized to $Q = 8$ bits with $n = 1$ flipped bit per perturbed parameter. Because the pretrained networks are highly vulnerable to single-bit flips (failing all pilot tests at $m = 1$), we fine-tune them using a hinge loss on the margin. To bound the perturbation intervals, we clip all weights and biases to $[-c, c]$ after each step. The clip bound c is fixed prior to verification as the smallest value that maintains accuracy within 3% of the pretrained network (§C).

Verification Instances. An verification instance pairs one input with $m \in \{1, 2, 4\}$ flipped parameters, evaluated under a 60-second timeout. The input perturbation radius is 0 or $2/255$ (0.01 for ACAS Xu). To ensure fair evaluations, we follow BFAVerifier by selecting the first 15 correctly classified test inputs that bound propagation successfully verifies at $m = 0$. For ACAS Xu, inputs are sampled from the normalized domain using the network's advisory as the label. No CIFAR-10 image passes this filter at radius $2/255$, so Tab. 1 has 13 network–radius pairs (585 instances), and two pretrained networks add 40 (§C).

Baselines We compare mBFV against BFAVerifier and an exact MILP extended to m flips. The MILP is the only complete method for $m \geq 2$, building a model with one set of binary variables per parameter; however, if model construction times out on the first instance of a run, all remaining instances for that run are reported as unproved (§C).

5.2 MBFV'S PERFORMANCE COMPARED TO BASELINES

Tab. 1 compares mBFV against an ablation without branch and bound (mBFV$_{\text{w/o BaB}}$), BFAVerifier (BFAV), and an exact MILP baseline. At $m = 1$, mBFV verifies 121 total instances, compared to 38 for BFAV and 0 for MILP. mBFV performs particularly well on the larger MNIST and CIFAR networks (269k–1.15M parameters), verifying 104 of 105 instances while both baselines time out.

However, BFAV verifies more instances than mBFV on the smaller ACAS Xu networks (13k parameters). Because these networks have relatively few parameters, BFAV's single parameter enumeration remains computationally feasible. Additionally, their large clip bound ($c = 0.5$) means single bit flips can substantially change outputs. BFAV handles this shift precisely with exact MILP, whereas mBFV's relaxations become too loose in this specific setting. Fig. 2 splits these results by network family and shows that branch and bound gains the most on the fully connected MNIST and CIFAR networks.

Tab. 1: Verified instances out of 15 inputs per network and radius within 60 s.

		$m=1$				$m=2$				$m=4$			
network	rad.	mBFV$_{\text{w/o BaB}}$	mBFV	BFAV	MILP	mBFV$_{\text{w/o BaB}}$	mBFV	BFAV	MILP	mBFV$_{\text{w/o BaB}}$	mBFV	BFAV	MILP
ACAS Xu 1_1	0	0	4	**5**	0	0	0	–	–	0	0	–	–
	0.01	0	4	**7**	0	0	0	–	–	0	0	–	–
ACAS Xu 2_1	0	0	3	**8**	0	0	0	–	–	0	0	–	–
	0.01	0	0	**8**	0	0	0	–	–	0	0	–	–
ACAS Xu 3_3	0	0	4	**5**	0	0	0	–	–	0	0	–	–
	0.01	0	2	**5**	0	0	0	–	–	0	0	–	–
MNIST FC 2×256	0	**15**	**15**	0	0	14	**15**	–	–	13	**14**	–	–
	2/255	**15**	**15**	0	0	**14**	**14**	–	–	5	**14**	–	–
MNIST FC 4×256	0	**15**	**15**	0	0	**15**	**15**	–	–	1	**15**	–	–
	2/255	**15**	**15**	0	0	5	**15**	–	–	0	**14**	–	–
MNIST FC 7×250	0	**15**	**15**	0	0	**15**	**15**	–	–	2	**13**	–	–
	2/255	**15**	**15**	0	0	5	**15**	–	–	0	**4**	–	–
CIFAR FC 7×250	0	5	**14**	0	0	0	**9**	–	–	0	0	–	–
Total		95	**121**	38	0	68	**98**	–	–	21	**74**	–	–

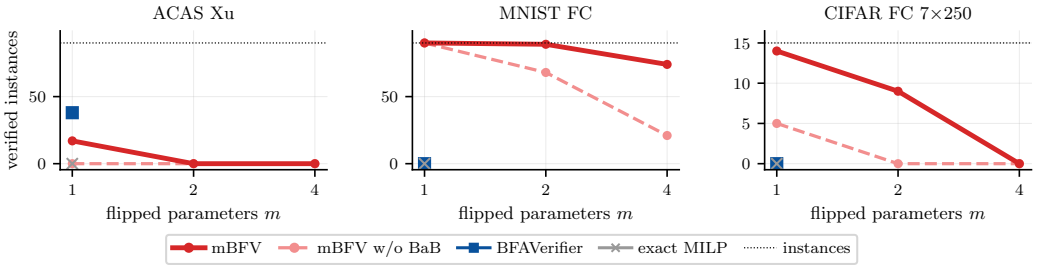

Fig. 2: Verified instances against the number m of flipped parameters per network family.

As illustrated in Fig. 3a, BFAV and MILP do not scale beyond a single flip, mBFV is the only method that is able to verify instances at $m = 2$ (98 verified) and $m = 4$ (74 verified). Without BaB, mBFV$_{\text{w/o BaB}}$'s performance decreases from 95 verified instances at $m = 1$ to 21 at $m = 4$. In contrast, mBFV partitions the search space to maintain tighter bounds, allowing it to verify more instances, *e.g.*, on MNIST FC 4×256 ($m = 4$, radius 0), mBFV verifies all 15 instances while mBFV$_{\text{w/o BaB}}$ verifies only 1. The exact MILP cannot verify any instances with $m \geq 2$.

5.3 Effectiveness of Branch and Bound

Fig. 3b presents verification runtimes across all numbers of flipped parameters. mBFV$_{\text{w/o BaB}}$ performs only a single bound propagation, which makes it fast but cannot verify many instances. mBFV matches mBFV$_{\text{w/o BaB}}$ on easy instances, verifying about 190 instances in under 0.1 seconds. For more complex problems, mBFV performs branching and computes additional bound propagations, which requires more runtime, scaling progressively up to the 60 second timeout. BFAVerifier requires higher base latency, requiring over 2.5 seconds even for its easiest instances, and verifies only 38 instances in total, while the exact MILP does not verify any within the time limit.

Fig. 3c illustrates the search space behavior by plotting the evaluated subproblems per verified instance. The vertical lines at 10^0 represents instances solved entirely at the root node by the continuous relaxation, matching the capabilities of mBFV$_{\text{w/o BaB}}$. At $m = 1$, this relaxation is tight enough to verify nearly 100 instances without branching. As the number of flipped parameters increases, multiple parameter flips widen the downstream bounds, reducing the number of instances verified at the root to 68 at $m = 2$ and 21 at $m = 4$. For instances that require search space partitioning, the necessary depth scales substantially with m. Verifying the most challenging inputs requires exploring fewer than 100 subproblems at $m = 1$, nearly 2000 at $m = 2$, and upwards of 4000 at $m = 4$.

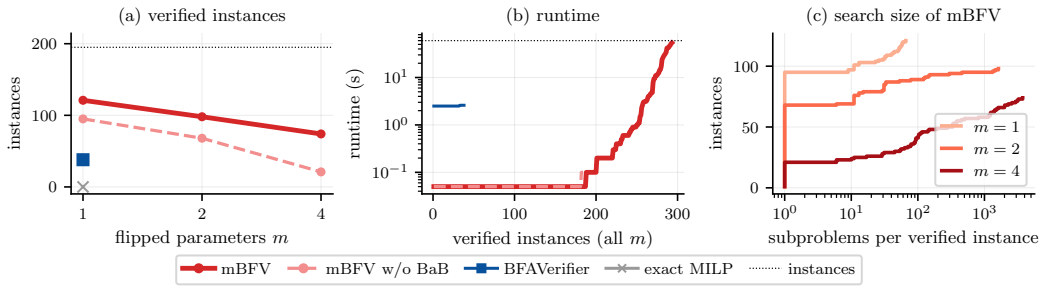

Fig. 3: Verified instances, runtime, and search size of mBFV over all networks.

6 Related Work

Bit-flip attacks and defenses. Existing defenses against bit-flip attacks primarily rely on empirical techniques, such as triplicating critical weights (Schmedding et al., 2024), planting honey neurons to attract and detect flips (Liu et al., 2023), or protecting weights with error-detecting codes (Velčický et al., 2026). However, these approaches do not provide formal verification certificates. Furthermore, in our theoretical setting, introducing such error-detecting codes would merely alter the reachable set R_n in Eq. 2 without fundamentally addressing the need for formal guarantees.

Verifying quantized networks. Exact verifiers for quantized networks typically assume fixed weights and verify robustness against input perturbations. These methods rely on bit-precise SMT encodings (Giacobbe et al., 2020; Henzinger et al., 2021), integer linear programming formulations like QVIP (Zhang et al., 2022) and EQV (Huang et al., 2024), or bounded model checking as in QNNVerifier (Song, 2024). Crucially, these frameworks target quantized activations, which differs from the BFAVerifier threat model adopted in our work. While BFAVerifier (Zhang et al., 2025) introduces a symbolic-parameter domain and a MILP formulation to certify network weights, it is strictly limited to verifying flips in a single parameter. In contrast, our approach scales to verify simultaneous bit flips across any m parameters.

Bound propagation and branch and bound. State-of-the-art verification frameworks, such as CROWN (Zhang et al., 2018; Xu et al., 2020; Wang et al., 2021) and DeepPoly (Singh et al., 2019), efficiently compute linear bounds via back-substitution. The integrality property underpinning Prop. 3.1 is standard in classical combinatorial optimization (Schrijver, 2003), and we leverage this property in every concretization step of our bound propagation. Finally, our second core contribution, branch and bound over perturbation locations, is orthogonal to existing activation-space branching methods. Consequently, it could be naturally combined with strategies like the ReLU branching employed in β-CROWN (Wang et al., 2021).

7 Conclusion

We presented mBFV to verify quantized networks against m-parameter bit-flip perturbations, surpassing the $m = 1$ limit of BFAVerifier. By capturing $\binom{K}{m}$ choices via a single multi-bitflip bound propagation and branching over perturbation locations, mBFV requires at most $1 + \binom{2\nu-L}{m}$ propagations, verifying 293 out of 625 problems.

The current limitations of mBFV include the lack of support for certain operations such as max pooling and residual connections. Additionally, we observe a tradeoff between accuracy and robustness, with accuracy decreasing from 98.1% to 96.2% on the MNIST fully connected 2×256 architecture. In future work, we plan to distribute the m bit-flips across multiple layers, enabling verification of perturbations spread over several layers by solving parallel single-layer subproblems with fewer flips. We also aim to incorporate tighter α-CROWN Xu et al. (2020) relaxations within each subproblem.

AI USE STATEMENT

In this work, we used generative AI tools (Claude) to assist in writing the Lean proofs that mechanize Prop. 3.1 (exactness and soundness of the top-m concretization), Prop. 4.1 (number of bound propagations), Prop. A.1, Prop. A.2, and Prop. A.3 (soundness of branch and bound), the steps of §3, the quantization facts of §2, the MILP encoding, and the worked example. We also used the same AI tools to implement mBFV by adapting previous reference implementation of NNV abstraction domains and BaB algorithms to the m bitflip verification problem.

We have not used generative AI tools for help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, design or provide feedback on research methodology or experiments, propose or refine hypotheses, interpret results, and the rest of the required disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools (Claude) to generate the Python plotting code for Fig. 3 and Fig. 2 and the TikZ code of the worked example in §A.1. We have reviewed all AI-assisted work, *e.g.*, we read and revise the Python code refactored by AI and develop unittest cases to validate them. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All authors have read and adhere to the ICLR 2027 Code of Ethics.

REPRODUCIBILITY STATEMENT

mBFV is available at `https://anonymous.4open.science/r/mbfv`.

REFERENCES

Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T. Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer*, 25(3):329–339, 2023.

Jianshuo Dong, Han Qiu, Yiming Li, Tianwei Zhang, Yuanjie Li, Zeqi Lai, Chao Zhang, and Shu-Tao Xia. One-bit flip is all you need: When bit-flip attack meets model training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Mirco Giacobbe, Thomas A. Henzinger, and Mathias Lechner. How many bits does it take to quantize your neural network? In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2020.

Thomas A. Henzinger, Mathias Lechner, and Đorđe Žikelić. Scalable verification of quantized neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraş. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *USENIX Security Symposium*, 2019.

Pei Huang, Haoze Wu, Yuting Yang, Ieva Daukantas, Min Wu, Yedi Zhang, and Clark Barrett. Towards efficient verification of quantized neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

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

Yoongu Kim et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014.

Xiang Li, Lannan Luo, and Qiang Zeng. Backdoor attacks on neural networks via one-bit flip. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

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

Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. NeuroPots: Realtime proactive defense against bit-flip attacks in neural networks. In *USENIX Security Symposium*, 2023.

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Anna Schmedding, Lishan Yang, Adwait Jog, and Evgenia Smirni. Aspis: Lightweight neural network protection against soft errors. In *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2024.

Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL), 2019.

Xidan Song. *Automated Verification and Repair of Quantized Neural Networks*. PhD thesis, University of Manchester, 2024.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Patrik Velčický, Jakub Breier, Mladen Kovačević, and Xiaolu Hou. DeepNcode: Encoding-based protection against bit-flip attacks on neural networks. *IEEE Transactions on Emerging Topics in Computing*, 14(3), 2026.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Fan Yao, Adnan Siraj Rakin, and Deliang Fan. DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *USENIX Security Symposium*, 2020.

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

Yedi Zhang, Zhe Zhao, Guangke Chen, Fu Song, Min Zhang, Taolue Chen, and Jun Sun. QVIP: An ILP-based formal verification approach for quantized neural networks. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022.

Yedi Zhang et al. BFAVerifier: Verifying bit-flip attack resistance of quantized neural networks. *arXiv preprint arXiv:2502.16286*, 2025.

A BRANCH AND BOUND: WORKED EXAMPLE AND SOUNDNESS

This appendix works out a five-neuron example (§A.1), proves that the subproblems cover every perturbation (§A.2), and proves Prop. 4.1 (§A.3).

A.1 WORKED EXAMPLE: THE SUBPROBLEMS COVER THE INTEGER PROGRAM

Neural Network. Fix the input and take one hidden layer of five neurons with the margin

$$\hat{z}_j = 1 + d_j \quad (j = 1, \dots, 5), \qquad z_j = \mathrm{ReLU}(\hat{z}_j), \qquad f = z_1 + z_2 + z_3 + z_4 + z_5 - \tfrac{5}{2} \tag{17}$$

Only the biases $b_1, \dots, b_5$ are perturbable, so neuron $\hat{z}_j$ has the single parameter b_j and nominal value 1. The change d_j of b_j lies in the change interval Eq. 3, here $[\ell, u] = [-3, 0.5]$, and at most $m = 2$ parameters flip. With more parameters per neuron the argument is the same, and only the ranges of the $\hat{z}_j$ grow.

Integer program. A perturbation selects $s \in \mathcal{S}_2 = \{s \in \{0,1\}^5 : \sum_j s_j \le 2\}$ and changes $d_j \in [-3s_j, 0.5s_j]$, so an unselected bias keeps $d_j = 0$. Problem (5) becomes the integer program

$$f^\star = \min_{s \in \mathcal{S}_2} F(s), \qquad F(s) = \min_{d_j \in [-3s_j,\, 0.5s_j]} \sum_j \mathrm{ReLU}(1 + d_j) - \tfrac{5}{2} = 5 - \textstyle\sum_j s_j - \tfrac{5}{2} \tag{18}$$

since a perturbed neuron drops to $\mathrm{ReLU}(1 - 3) = 0$ and an unperturbed neuron stays at 1. The ten pairs are the worst of the 16 selections, so $f^\star = 0.5 > 0$ and the instance is verified.

Bounding. BOUND(S) is multi-bitflip bound propagation with only the biases in $U = \bigcup S$ perturbable. For $j \notin U$ the range of $\hat{z}_j$ is $[1, 1]$, so $z_j = 1$ exactly. For $j \in U$ the range $[-2, 1.5]$ makes the neuron unstable, and CROWN takes the lower line $z_j \ge 0$ since the upper line has slope $1.5/3.5 < 0.5$. BOUND(S) is therefore the optimum of the linear program

$$R(U) \;=\; \min \sum_j z_j - \tfrac{5}{2} \quad \text{s.t.} \quad z_j \ge 0 \;(j \in U), \qquad z_j = 1 \;(j \notin U) \tag{19}$$

Hence $R(U) = 5 - |U| - \frac{5}{2}$, so a subproblem is proved exactly when it has at most two neurons.

At the root, where all five neurons are perturbable, $R(\{1, \dots, 5\}) = -2.5 < 0$ although $f^\star = 0.5$. In the integer program (18), $z_j = 0$ needs $s_j = 1$, so at most two neurons drop, whereas the linear program (19) sets all five z_j to 0 at once. This gap is the loss described in §4, where the range of each neuron is computed as if it alone were perturbed. The selection s and the changes d enter only through $\hat{z}_j$, which the line $z_j \ge 0$ does not involve, so the constraint $\sum_j s_j \le 2$ never reaches it. Every term g_j is therefore 0, and the top-2 selection adds nothing. Prop. 3.1 makes the selection exact given the terms, whereas the loss lies in the ReLU lines.

Branching. For a set of neurons U, let $\mathcal{S}_2(U)$ be the selections with $s_j = 0$ for $j \notin U$. Every perturbation inside U is feasible for (19), since $z_j = \mathrm{ReLU}(\hat{z}_j) \ge 0$, so

$$R(U) \;\le\; \min_{s \in \mathcal{S}_2(U)} F(s) \tag{20}$$

The neurons $U_1, \dots, U_q$ of the subproblems *cover* the integer program if $\mathcal{S}_2 = \mathcal{S}_2(U_1) \cup \cdots \cup \mathcal{S}_2(U_q)$. For a cover, taking the minimum of (18) over each $\mathcal{S}_2(U_t)$ and applying (20) to each gives

$$f^\star \;=\; \min_{t \le q} \; \min_{s \in \mathcal{S}_2(U_t)} F(s) \;\ge\; \min_{t \le q} R(U_t) \tag{21}$$

Proving $R(U_t) > 0$ for every t thus proves $f^\star > 0$, even if the sets $\mathcal{S}_2(U_t)$ overlap. Each subproblem is a smaller linear program with fewer relaxed ReLUs. By Prop. A.1, the children that BRANCH (15) returns cover their parent.

The search. The search proves the instance at depth 3 after 16 bound propagations (Fig. 4). Each subproblem shows its groups above its bound R (19), in red if it fails, in green if it is proved, and dashed if it repeats an earlier one and is not bounded again. Gray boxes show how BRANCH (15) splits the largest group until T has more than m groups (1) and returns every set of m groups of T as a child (2). The network has one perturbable layer, so the root is the single group $\{1, \dots, 5\}$ and is branched directly. Its first split gives $|T| = 2 = m$, so BRANCH also splits $\{3, 4, 5\}$. The neuron sets $\{1, 2, 3\}$, $\{1, 2, 4, 5\}$, and $\{3, 4, 5\}$ of the three depth-1 subproblems contain all ten pairs, so they cover the integer program. Every pair subproblem has $R = 0.5 > 0$ and is proved, and $\{\{4\}, \{5\}\}$ is generated twice but bounded once. All ten pairs are then proved and the answer is verified, with the bound $0.5 = f^\star$ by (21). The pair subproblems are exact here because they equal the worst selections. In a deeper network, a pair subproblem still relaxes the ReLUs of its two neurons and of later layers, so its bound can stay below the integer optimum.

Remark 1 (Lean mechanization of the worked example)**.** *The worked example is mechanized in* `Proof/Example.lean`*, with the five neurons indexed by* `Fin 5`*.* `F_isLeast`*,* `fstar_isLeast`*, and* `R_eq` *prove* $F(s) = 5 - |s| - \frac{5}{2}$ *in* (18)*,* $f^\star = 0.5$*, and the optimum* $R(U)$ *of* (19)*.* `proved_iff` *proves that a subproblem is proved exactly when it has at most two neurons.* `ex_sound` *proves* (20) *assuming only* $s \subseteq U$*, and* `ex_cover` *proves* (21) *for any finite nonempty family of subproblems that covers the selections.* `depth1_cover` *and* `tab_ex_cover` *check that the depth-1 subproblems cover the ten pairs, and* `fstar_cover` *proves that the ten pair subproblems give the bound* $0.5 = f^\star$*. The proofs mainly use* `decide` *on the finite set of selections,* `rw`*, and* `norm_num`*, with* `linarith` *closing the inequalities. They contain no* `sorry` *and introduce no axioms beyond Lean's standard* `propext`*,* `Classical.choice`*, and* `Quot.sound`*.*

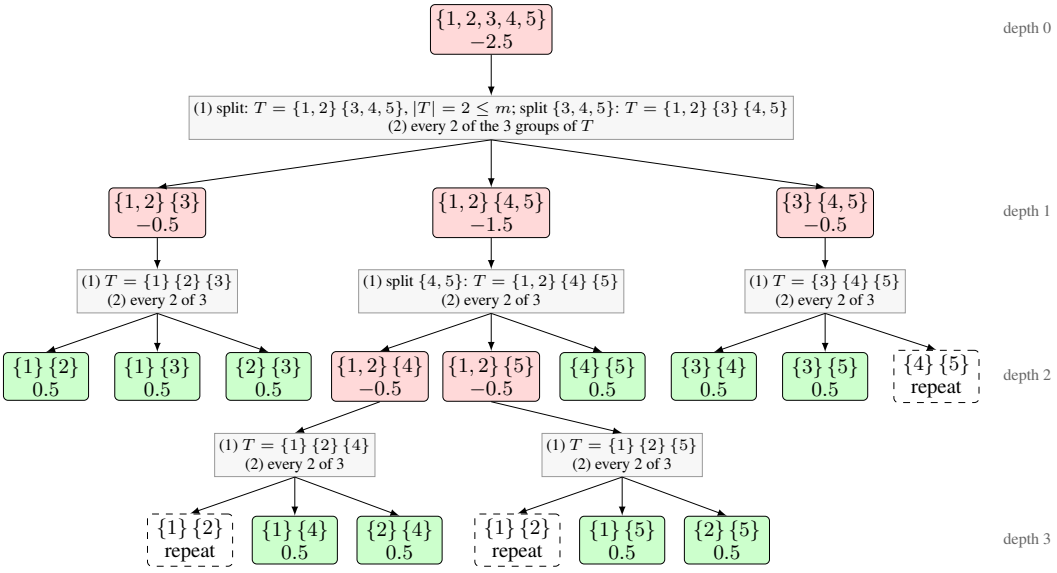

Fig. 4: Branch and bound on the five-neuron example.

A.2 WHY THE COVER IS SOUND

A perturbation ρ *lies in* a subproblem S when every parameter it changes belongs to $\bigcup S$. Soundness of the verdict verified rests on one invariant, which the branching maintains.

Proposition A.1 (Children cover their parent)**.** *Let S be a subproblem, $G \in S$ a group, and $G = G_1 \uplus G_2$ a split of G into two disjoint non-empty parts, so that $T = (S \setminus \{G\}) \cup \{G_1, G_2\}$ has pairwise disjoint groups and $\bigcup T = \bigcup S$. If $|T| > m$, every (m, n)-perturbation that lies in S lies in one of the children $\{C \subseteq T : |C| = m\}$.*

Proof. Let ρ lie in S and let $D \subseteq T$ be the groups that contain a parameter changed by ρ. The groups of T are pairwise disjoint, so each changed parameter lies in exactly one group of T. Hence $|D| \leq$

m, because ρ changes at most m parameters. Since $|T| \geq m$, choose any C with $D \subseteq C \subseteq T$ and $|C| = m$. This C is one of the children, and every parameter changed by ρ lies in $\bigcup D \subseteq \bigcup C$. □

Remark 2 (Lean mechanization of Prop. A.1 (Children cover their parent)). `children_cover` *in* `Proof/Cover.lean` *proves Prop. A.1 assuming only* $m \leq |T|$ *and that the parameters changed by* ρ *lie in* $\bigcup T$. *It picks one group of* T *for each changed parameter and extends these at most* m *groups to* m *groups of* T. `cover_step` *restates it for the split* T, *and* `child_ssubset` *proves that a child of pairwise disjoint non-empty groups allows strictly fewer parameters than its parent. The proofs mainly use* `obtain`, `rw`, *and induction over finite sets, contain no* `sorry`, *and add no axioms beyond* `propext`, `Classical.choice`, *and* `Quot.sound`.

If $|T| \leq m$, the algorithm splits again before bounding anything. Each split keeps the groups disjoint and $\bigcup T = \bigcup S$, and it adds one group to T. After finitely many splits, either $|T| > m$ and Prop. A.1 applies, or every group is a single neuron and the subproblem is a leaf.

Proposition A.2 (Cover invariant). *At the start of every iteration of Alg. 2, every* (m, n)*-perturbation lies in some subproblem of* $Q \cup P$.

Proof. Let $\mathcal{G}$ be the set of subproblems generated so far, the initial subproblems and all children. We prove a stronger invariant: every (m, n)-perturbation that lies in some $H \in \mathcal{G}$ lies in a subproblem of P, or in a subproblem $S \in Q$ with $\bigcup S \subseteq \bigcup H$. *Initially*, $\mathcal{G} = Q$, and H itself is such a subproblem. The layers are pairwise disjoint and cover all K parameters, so the argument of Prop. A.1 with $T = \{\text{layers}\}$ puts every perturbation in an initial subproblem. If $L \leq m$, the initial subproblems are those of BRANCH(all layers), which cover every perturbation by the same proposition. The invariant therefore implies the proposition. *Step*: a subproblem S leaves Q in exactly three ways. (i) $\text{BOUND}(S) > 0$, and S enters P. (ii) The neurons of S lie inside those of a proved $S' \in P$, so $\bigcup S \subseteq \bigcup S'$ and what lies in S lies in S'. (iii) $\text{BOUND}(S) \leq 0$, and S is replaced by its children, which cover S by Prop. A.1, and each child C satisfies $\bigcup C \subsetneq \bigcup S$ because the groups are disjoint and non-empty. A perturbation that lay in S lies in some child C, which is queued if it is new. If C was generated before, the invariant for C gives a proved subproblem, or a queued subproblem inside $\bigcup C \subsetneq \bigcup S$, which is therefore not S and stays queued. □

Remark 3 (Lean mechanization of Prop. A.2 (Cover invariant)). `cover_invariant` *in* `Proof/Cover.lean` *proves Prop. A.2 assuming only that the initial subproblems cover every perturbation of at most* m *parameters (*`init_cover`*). The search is modeled by the steps (i) to (iii), where a branch step queues only the children not generated before, and* `branch_step` *proves that* BRANCH *on pairwise disjoint non-empty groups is such a step.* `inv_step` *and* `inv_reach` *prove that every step preserves the invariant of the proof above, which thus holds in every reachable state. The proofs mainly use* `obtain`, `by_cases`, *and induction over reachable states, contain no* `sorry`, *and add no axioms beyond* `propext`, `Classical.choice`, *and* `Quot.sound`.

Proposition A.3 (Soundness of the search). *If Alg. 2 returns verified, then* $f^\star > 0$, *and* $\min_{S \in P} \text{BOUND}(S)$ *is a lower bound on* $f^\star$.

Proof. The algorithm returns verified only with $Q = \emptyset$, so by Prop. A.2 every (m, n)-perturbation ρ lies in some $S \in P$. $\text{BOUND}(S)$ is multi-bitflip bound propagation with only $\bigcup S$ perturbable with at most m flipped parameters. By §3, $\text{BOUND}(S)$ is a lower bound on $f^{\rho'}(x)$ for every $x \in \mathcal{B}$ and every (m, n)-perturbation ρ' lying in S, and in particular for ρ. Since $\text{BOUND}(S) > 0$ for every $S \in P$, the claim follows. □

Remark 4 (Lean mechanization of Prop. A.3 (Soundness of the search)). `search_sound` *and* `search_lower_bound` *in* `Proof/Cover.lean` *prove Prop. A.3 from* `cover_invariant`, *assuming covering initial subproblems, a stop with* $Q = \emptyset$, *and that* $\text{BOUND}(S)$ *bounds the margin from below under every perturbation that lies in* S, *which is the soundness of multi-bitflip bound propagation (7).* `search_sound` *adds* $\text{BOUND}(S) > 0$ *for* $S \in P$ *and concludes a positive margin, and* `search_lower_bound` *adds* $P \neq \emptyset$ *and concludes the lower bound* $\min_{S \in P} \text{BOUND}(S)$. *The proofs mainly use* `obtain`, *contain no* `sorry`, *and add no axioms beyond* `propext`, `Classical.choice`, *and* `Quot.sound`.

Nothing is claimed for the other two verdicts: unknown reports the bound of the failing leaf, and timeout reports the bound of the root, which holds with all K parameters perturbable.

A ReLU phase split (Wang et al., 2021) *partitions* its parent, whereas our children *cover* it, since a perturbation of fewer than m parameters lies in several of them. A leaf of fixed phases is exact, but a leaf of single neurons is not, because its ReLU lines can still be loose.

Refinement. A child allows a subset of its parent's parameters, yet its bound is occasionally worse, since CROWN's lower ReLU line (the slope 0 or 1 nearest the upper line) can flip when a recomputed neuron range tightens. A repository script compares 5,124 parent–child pairs from eight networks (three refinement levels, $m \in \{1, 2, 3\}$, at a point and on a box), and in every pair the child's parameter set is a subset of its parent's. In 63 pairs (1.2%), the child's bound is worse, and the largest drop is 4.9. This costs bound propagations, never soundness, because only failing subproblems are split, no parent bound is reused, and Prop. A.3 takes the minimum over independently proved subproblems.

A repository test drives the search to the leaves with a stub that proves exactly the subproblems of at most m neurons (or channels), and checks on dense and convolutional networks that *every* set of at most m neurons lies in a proved subproblem. Another test compares $\textsc{Bound}(S)$ at a point input with the exact minimum over all perturbations of at most m parameters in $\bigcup S$, and over 32 (subproblem, network) pairs no bound exceeds it. Of these bounds, 4 are tight to machine precision, so on those subproblems the multi-parameter concretization is exact. §5 adds the end-to-end check that no verified answer contradicts a counterexample of a baseline.

A.3 Proof of Prop. 4.1

Proof. Upper bound. The *splitting tree* of layer i has the whole layer as its root, and every group of at least two neurons has its two parts as children. Its leaves are the ν_i single neurons and every inner node has two children, so it has $2\nu_i - 1$ nodes, and the L trees together have $2\nu - L$ nodes. Every group of the search is such a node, since the first groups are whole layers and Branch replaces a node by its children. Every subproblem bounded after the root is a set of exactly m nodes, since the first subproblems are sets of m layers (if $L \le m$, those of Branch(all layers)) and Branch returns only subsets of m groups of T. A subproblem already generated is not queued again, so each set of m nodes is bounded at most once. Therefore, at most $\binom{2\nu-L}{m}$ subproblems are bounded after the root, and the root adds one. *Lower bound.* Since $\nu > m$, the root fails. A subproblem bounded after the root has m non-empty groups, so it is proved exactly when every group is a single neuron, and a failing subproblem has a group of at least two neurons. One split gives $|T| = m + 1$, so Branch returns children. If $L \le m$, the first Branch splits the layers, and as long as $|T| \le m < \nu$ some group has at least two neurons, so it also reaches $|T| = m + 1$. The search thus never returns unknown. There is no time limit, and by the upper bound only finitely many subproblems are queued, each once, so Q empties and the search returns verified. By Prop. A.2, the proved subproblems then cover every perturbation. A perturbation that changes one parameter in each of m given neurons lies only in the proved subproblem made of exactly these neurons, so all $\binom{\nu}{m}$ such subproblems are bounded. For $m = 1$, a failing subproblem is one node, and its children are the node's two parts. No node is skipped, since the proved subproblems are single neurons, so every node is bounded once. With $L \ge 2$, the first subproblems are the L layers, so the search makes exactly $1 + 2\nu - L$ bound propagations. □

For a fixed m, both bounds grow as ν^m, and for $\nu \gg m$ they differ by a factor of about $(2 - L/\nu)^m$, close to 2^m when the layers are wide. A repository test runs the search with the stub of §A.2 on five networks for $m \le 3$ and checks both bounds and the equality for $m = 1$.

Remark 5 (Lean mechanization of Prop. 4.1 (Number of bound propagations))**.** *The upper bound is mechanized in* `Proof/Subproblems.lean`*, assuming only* $\nu_i \ge 1$*.* `card_splitTree`*,* `splits_mem`*, and* `card_groupUniv_le` *prove that the splitting tree of a layer, built as in the implementation, has* $2\nu_i - 1$ *nodes and is closed under splitting, and that the* L *trees have at most* $2\nu - L$ *nodes.* `subproblems_card_le` *then bounds the number of distinct subproblems generated by* Branch *by* $\binom{2\nu-L}{m}$*. The counting step of the lower bound is* `stub_proves_m_sets` *and* `stub_card_ge` *in* `Proof/Cover.lean`*. When the search stops with* $Q = \emptyset$ *and every proved subproblem has at most* m *neurons, every set of* m *neurons is the neuron set of a proved subproblem, so at least* $\binom{\nu}{m}$ *subproblems are proved. The implementation supplies the remaining facts: no subproblem is queued twice, and the search terminates. The proofs mainly use* `omega`*,*

`rw`, `obtain`, and functional induction on the splitting tree, contain no `sorry`, and add no axioms beyond `propext`, `Classical.choice`, and `Quot.sound`.

B Proof of Prop. 3.1 and Lean Mechanizations

Proof. Step 1. Pick $\theta \le 0$ with $g_p \le \theta$ for $p \in T$ and $g_p \ge \theta$ for $p \notin T$. If some parameter lies outside T, then $\theta = \min\big(0, \min_{q \notin T} g_q\big)$ works, since $g_p \le \min(0, g_q)$ for $p \in T$ and $q \notin T$. Otherwise $\theta = 0$ works. *Step 2.* For every $s \in \mathcal{C}_m$ and every p,

$$(s_p - \mathbf{1}_T(p))\, g_p \;\ge\; (s_p - \mathbf{1}_T(p))\, \theta \tag{22}$$

because $s_p - 1 \le 0$ and $g_p \le \theta$ for $p \in T$, and $s_p \ge 0$ and $g_p \ge \theta$ for $p \notin T$. Summing over p gives

$$\sum_p s_p g_p - \sum_{p \in T} g_p \;\ge\; \theta\Big(\sum_p s_p - |T|\Big) \tag{23}$$

Step 3. The right-hand side is ≥ 0, which proves (a). If some parameter lies outside T, then $|T| < K$, so $|T| = \min(m, K) = m$, and both θ and $\sum_p s_p - |T|$ are ≤ 0. Otherwise $\theta = 0$. *Step 4.* The 0/1 vector $\mathbf{1}_T$ has $\sum_p \mathbf{1}_T(p) = |T| \le m$, so it is in $\mathcal{S}_m \subseteq \mathcal{C}_m$ with objective $\sum_p \mathbf{1}_T(p)\, g_p = \sum_{p \in T} g_p$, which proves (b). By (a), (b), and $\mathcal{S}_m \subseteq \mathcal{C}_m$, the three minima are equal. □

Remark 6 (Lean mechanization of Prop. 3.1)**.** *Prop. 3.1 is mechanized in Lean 4 with Mathlib as `topm_le_lp` (part (a)) and `lp_relaxation_exact` (the three equal minima) in `Proof/TopM.lean`, whose only hypothesis on the terms is $g_p \le 0$. They characterize T by $|T| = \min(m, K)$ and $g_p \le g_q$ for $p \in T$ and $q \notin T$, so no sorting is needed, and `exists_topm` constructs such a T for every g. The threshold θ is a dual certificate (`le_of_certificate`) reducing the proof to elementwise linear inequalities. `topm_sound` proves the soundness step of §3.3, that a perturbation of at most m parameters whose terms are at least g_p changes the bound by at least $\mathrm{top}_m(g)$. The proofs mainly use `rw`, `refine`, and `linarith`, contain no `sorry`, and add no axioms beyond `propext`, `Classical.choice`, and `Quot.sound`.*

Remark 7 (Lean mechanization of the soundness of multi-bitflip bound propagation)**.** *In `Proof/Terms.lean`, a corner attains the minimum in* (9) *over any nonempty rectangle (`corner_isLeast`), $\underline{g}_p \le 0$ follows from $\ell_p \le 0 \le u_p$ alone (`corner_nonpos`), and minimizing a term linear in the change over the interval hull of R_n gives its minimum over R_n (`hull_min_isLeast`). `sep_isLeast`, `concretize_sound`, and `concretize_exact` prove with `topm_sound` that $\min_{x \in \mathcal{B}}(Ax + c) + \mathrm{top}_m(g)$ is a lower bound on the margin under every flip in at most m parameters and the minimum of the relaxed program. For a convolution, `sum_min_attained` proves that the range of S_p is the sum of the ranges of its summands, and `conv_corner_le` applies the corner bound to $D_p S_p$. The proofs mainly use `rcases`, `refine`, and `nlinarith`, contain no `sorry`, and add no axioms beyond `propext`, `Classical.choice`, and `Quot.sound`.*

Remark 8 (Lean mechanization of the quantization facts)**.** *In `Proof/Quant.lean`, `card_reach_real` proves that R_n has exactly $\sum_{r \le n} \binom{Q}{r}$ elements, assuming only $\Delta_i \ne 0$, and `change_interval_contains_zero` proves $\ell_\alpha \le 0 \le u_\alpha$ in* (3)*. `sign_flip_min_real` proves that one sign-bit flip reaches $-2^{Q-1}\Delta_i$, and `clip_change_bound` proves the bound $|D^{(i)}_{jk}| \le (2^Q - 1)\, c/(2^{Q-1} - 1)$ used in fine-tuning (§C), assuming only that every real parameter of the layer lies in $[-c, c]$. The proofs mainly use `rw`, `simp`, and `refine`, contain no `sorry`, and add no axioms beyond `propext`, `Classical.choice`, and `Quot.sound`.*

C Experimental Details

Networks. The architectures (Tab. 2) and ONNX files come from the VNN-COMP benchmarks `acasxu`, `mnistfc`, and `cora`. Our loader matches `onnxruntime` on 64 random inputs to a relative error below 10^{-6}, the float32 precision. The `cora` networks read images with swapped spatial axes, MNIST as (W, H) and CIFAR as (C, W, H). With this layout, their accuracy is 98.2% and 51.3%, respectively, against 13.6% and 20.6% with the usual layout. For ACAS Xu, accuracy is the agreement with the original network's advisory on sampled inputs. After plain 8-bit quantization, the published networks agree with only 7.9% to 58.2% of these advisories (pretrained column), so BFAVerifier also retrained them.

network	layers	K	clip c	Q-bit accuracy	pretrained
ACAS Xu 1_1	6×50	13,305	0.5	94.4%	7.9%
ACAS Xu 2_1	6×50	13,305	0.5	90.0%	58.2%
ACAS Xu 3_3	6×50	13,305	0.5	94.3%	17.9%
MNIST FC 2×256	2×256	269,322	0.1	96.2%	98.1%
MNIST FC 4×256	4×256	400,906	0.1	97.9%	97.7%
MNIST FC 7×250	7×250	575,260	0.05	97.6%	98.2%
CIFAR FC 7×250	7×250	1,147,260	0.05	51.9%	51.3%

Tab. 2: The evaluated networks.

Fine-tuning. Fine-tuning starts from the pretrained weights and minimizes the hinge loss $\max(0, M - \text{margin})$ with $M = 5$, batch size 128, and 10 epochs of Adam, clipping every parameter to $[-c, c]$ after each step. ACAS Xu has no training data, so its labels are the original network's advisories on 200k inputs sampled uniformly from its normalized domain. Before any verification run, c is fixed as the smallest value whose Q-bit accuracy is within 3 points of the pretrained network. The grids are $c \in \{0.05, 0.1, 0.2\}$ for the image networks and $\{0.5, 1, 2\}$ for ACAS Xu, each with learning rates 10^{-3} and 10^{-4}, and are extended upward when no value qualifies. For ACAS Xu no such c survives a single flip, so we take the most accurate setting whose three pilot instances at $m = 1$ are not all falsified (a counterexample checked by an exact forward pass). Every intermediate result is in the repository's logs.

Clipping bounds every change interval uniformly. With every real parameter in $[-c, c]$, the step is $\Delta_i \le c/(2^{Q-1} - 1)$, and flipping any bits of a Q-bit code changes the integer by at most $2^Q - 1$. Hence $|D_{jk}^{(i)}| \le (2^Q - 1)\, c/(2^{Q-1} - 1) \approx 2c$ for every parameter, whatever its value and whichever bits flip, including the sign bit.

Inputs. For the image networks, we permute the test set with seed 0, take the first 15 correctly classified images that bound propagation proves at $m = 0$, and clip their ℓ_∞ boxes of radius 0 or $2/255$ to $[0, 1]$. At radius $2/255$, no CIFAR-10 test image passes this filter (4,809 are misclassified and 5,191 are not proved at $m = 0$), so CIFAR-10 has instances only at radius 0. ACAS Xu inputs are sampled uniformly from the normalized domain, with radius 0 or 0.01 and the same $m = 0$ filter.

The standard (pretrained) networks. With the published weights, no method proves any instance of MNIST FC 2×256 or ACAS Xu 1_1 (radius 0, $m \in \{1, 2\}$, 10 inputs, 40 instances in total). mBFV returns unknown, and both baselines time out or find a counterexample, which motivates fine-tuning (§5).

Hardware and software. All runs use one machine with an Intel Core i9-9900K CPU (8 cores), 62 GB of memory, and an NVIDIA RTX 2080 Ti GPU (11 GB). Multi-parameter bound propagation and BFAVerifier's oracle run on the GPU in 64-bit floating point, and the MILP on the CPU with Gurobi 13 and 8 threads. Every instance has a limit of 60 s, and timed jobs run one at a time.

Implementation. We keep only a running top-m of the terms of each neuron and bound all neurons of a layer together as one matrix, with convolutions as in §C.2. Branch and bound bounds a batch of subproblems in one GPU call, each as a 0/1 mask over the K parameters. In a dense layer, the terms $\underline{g}_p$ of the weights of one neuron $\hat{z}_j$ differ only by the factor $\underline{A}_j$, so only the m most extreme terms of each neuron can enter the top-m. Keeping them instead of the full $(S, \text{out}, \text{in})$ tensor makes the 7×250 networks practical and leaves every bound bit-identical, as a test checks.

C.1 The Exact MILP Baseline

The baseline encodes the verification problem (5) exactly for at most m flipped parameters. Each perturbable parameter p has binaries $\delta_{p,r}$ with $\sum_r \delta_{p,r} = 1$, one per reachable value $v_{p,r} \in R_n(p)$ of (2), with $v_{p,0}$ nominal. The constraint $\sum_p (1 - \delta_{p,0}) \le m$ limits the number of flipped parameters. The product of a perturbed weight $W_{jk}^{(i)}$ and its input $z_k \in [\underline{z}_k, \overline{z}_k]$ is linearized with continuous y_r,

$\underline{z}_k\,\delta_{p,r} \le y_r \le \overline{z}_k\,\delta_{p,r}$ and $\sum_r y_r = z_k$. It contributes $\sum_r v_{p,r}\,y_r$ to $\hat{z}_j$, which is exact because exactly one $\delta_{p,r}$ is 1. A perturbed bias adds $\sum_r (v_{p,r} - v_{p,0})\,\delta_{p,r}$ to its nominal value. The copies of a kernel entry have separate products but share its binaries $\delta_{p,\cdot}$, so the constraint counts the entry once. Average pooling and other fixed linear maps enter as linear equalities, and the input ranges over the box $\mathcal{B}$. A ReLU with input range $[l, u]$ is encoded exactly as $z = \hat{z}$ if $l \ge 0$ and as $z = 0$ if $u \le 0$. Otherwise, a binary β enforces $z \ge \hat{z}$, $z \ge 0$, $z \le u\beta$, and $z \le \hat{z} - l(1-\beta)$. The bounds $[l, u]$ come from multi-bitflip bound propagation for the same m, so they hold for every perturbation.

For each margin row c of C, Gurobi minimizes $c^\top N^\rho(x)$ and stops once its best bound exceeds 10^{-6}, which proves the row, or once it finds a solution below -10^{-6}. The instance is verified only when the solver proves every row, and the rows share the time limit, including model building. With one set of binaries per parameter and one product per parameter occurrence, the model does not build within the limit for hundreds of thousands of parameters. If the first $m = 1$ instance of a run ends during model building, the remaining MILP instances of the run are skipped and reported as unproved.

Our re-implementation of BFAVerifier (Zhang et al., 2025) follows its Algorithm 1. For every parameter, the procedure BFA_RA splits the sorted reachable values into non-negative and negative parts and bisects each with a bound-propagation oracle in which the parameter is symbolic. The parameters that BFA_RA cannot prove go to the exact MILP with $m = 1$, and the GPU oracle answers the first query of every parameter in batches. For a convolution, as in Section 4.5.2 of that paper, the kernel entry is symbolic in all its copies, and each copy gets a separate linear bound. On 18 small dense networks and 36 small convolutional instances, both baselines agree with the ground truth from enumerating every perturbation.

Remark 9 (Lean mechanization of the exact MILP encoding)**.** *In* `Proof/Milp.lean`*,* `relu_exact` *proves that the big-M constraints hold for some binary β if and only if $z = \max(\hat{z}, 0)$, assuming only $l \le \hat{z} \le u$. For binaries in $\{0,1\}$ that satisfy the product constraints,* `onehot_product` *proves that the product contributes exactly $v_{p,r} z_k$ for the selected value, since every other y_r is 0.* `onehot_feasible` *proves that every value is feasible when $\underline{z}_k \le z_k \le \overline{z}_k$.* `budget_iff` *proves that $\sum_p (1 - \delta_{p,0}) \le m$ holds if and only if at most m parameters are perturbed, assuming only $\delta_{p,0} \in \{0,1\}$. The proofs mainly use* `rw`*,* `simp`*, and* `linarith`*, contain no* `sorry`*, and add no axioms beyond* `propext`*,* `Classical.choice`*, and* `Quot.sound`*.*

C.2 Convolutions and Pooling

A convolution is stored as its kernel with a map from every nonzero entry of the unrolled matrix to the kernel entry it copies, and back-substitution uses a transposed convolution. The term of kernel entry p sums the interval of $A_{j_e} z_{k_e}$ over its copies and takes the four corners of §3. An unfold followed by a matrix product computes these sums for all bounds at once. In the first layer, neuron (c, q) at position q of channel c depends only on the kernel entries and bias of channel c. Its range directly adds the m most extreme of their terms, which gives the same range as back-substitution with $A = I$ without forming the identity. Average pooling and other fixed linear maps between a ReLU and the next layer have no parameters, so they map ranges by their interval image and enter back-substitution as sparse matrices.

Sharing is essential for soundness. Take a network with a $1\times4\times4$ input, a 2×2 convolution with two channels, average pooling, and a dense layer ($Q = 8$, $n = 1$, one input point). At $m = 1$, treating the copies of each kernel entry as separate parameters gives a lower bound of $+4.94$ on a margin whose true worst value under one kernel flip is $+4.15$. Our bound is $+3.63$. A single flipped kernel entry changes all of its copies at once, and the separate-copies model counts this change as several perturbed parameters.