Annales Mathematicae et Informaticae
61 (2025) pp. 171–185
DOI: 10.33039/ami.2025.10.002
URL: https://ami.uni-eszterhazy.hu

Soft voting robustness in neural network ensembles with empirical analysis and formal verification

Ádám Kovács[ab], Roland Gunics[a], Gergely Kovásznai[a], Tibor Tajti[ab]

[a]Eszterházy Károly Catholic University
kovacs2.adam@uni-eszterhazy.hu
gunics.roland@uni-eszterhazy.hu
kovasznai.gergely@uni-eszterhazy.hu
tajti.tibor@uni-eszterhazy.hu

[b]University of Debrecen, Doctoral School of Informatics

Abstract. Neural network ensembles with soft voting improve accuracy and stability by aggregating multiple models; however, their reliability under individual model failure remains a critical concern. This paper addresses the robustness of soft-voting ensembles in safety-critical settings by combining empirical analysis and formal verification. We evaluate the impact of single-model failures on ensemble performance and find that soft voting yields graceful degradation, with only minimal loss in accuracy when one component model is removed or corrupted. In parallel, we develop a formal verification framework to investigate whether the ensemble's final prediction remains unchanged under any single-model failure scenario. The results demonstrate that soft-voting ensembles can maintain reliable outputs despite individual model failures, providing both empirical evidence and provable guarantees of fault tolerance in neural network ensembles.

Keywords: neural network, ensemble, robustness, model failure, formal verification, SMT

AMS Subject Classification: 68T07, 68T27, 68T37

1. Introduction

Modern safety-critical applications of AI, such as autonomous driving and healthcare, demand not only high accuracy but also *formal robustness guarantees* for

Accepted: October 8, 2025
Published online: October 28, 2025

reliability under all conditions. Ensemble learning is a proven approach to improve reliability: by aggregating multiple neural network models, ensembles achieve higher accuracy and greater stability than individual networks. However, empirical robustness (e.g., against noisy or adversarial inputs) achieved via techniques like adversarial training or data augmentation does not automatically translate into formal guarantees of correctness.

In high-stakes domains, we require provable assurance that the system will maintain correct operation even when some components fail or behave unexpectedly. This work introduces the concept of *voting robustness* as a measure of an ensemble's tolerance to individual model failures, and provides both empirical analysis and formally verified guarantees for this property.

We present a comprehensive study of voting robustness in neural network ensembles, evaluating their resilience across diverse architectures on a digit classification task. Our experiments demonstrate that even if one model in the ensemble is corrupted or fails entirely, the aggregated prediction remains essentially unchanged, with minimal loss in accuracy.

Beyond empirical evaluation, we provide formal verification of these ensembles using neural network verification tools based on Satisfiability Modulo Theories (SMT). By encoding the soft-voting mechanism into a single verifiable model, we can prove if the ensemble's prediction remains stable under single-model failures for any set of inputs. This dual approach of empirical analysis and formal verification establishes ensemble voting as a powerful mechanism for achieving built-in robustness, paving the way for AI systems that can be deployed with certified reliability in safety-critical domains.

2. Background and related work

2.1. Ensemble learning and voting schemes

Ensemble methods combine multiple learners to improve generalization performance. Classic ensemble techniques such as bagging, boosting, and random forests leverage model diversity to reduce variance and increase accuracy. In neural network ensembles, each model (potentially differing in initialization, architecture, or training data) contributes to the final prediction.

Voting can be performed in two primary ways: *majority voting* (hard voting), where the class with the most votes is selected, and *averaging* (soft voting), where the models' output probabilities are averaged and the class with the highest mean probability is chosen. Soft voting typically provides smoother decision boundaries and is easier to integrate into verification pipelines, as the voting layer is differentiable.

Prior work has demonstrated that ensembles can enhance adversarial robustness, as diversity among models makes it more difficult for a single adversarial example to mislead the entire ensemble. For instance, it was demonstrated that promoting diversity in non-maximal predictions via an adaptive regularizer en-

hanced both ensemble robustness and transfer resilience [10]. While substantial work has focused on empirical defenses such as adversarial training, noise resilience, and robustness to distributional shifts, much of this remains heuristic and lacks formal guarantees [6, 12]. Gross et al. showed that ensemble robustness verification is NP-hard and proposed SMT- and MILP-based encodings to either find optimal randomized attacks or formally prove robustness [5].

2.2. Voting robustness

We define *voting robustness* as the minimum number of model predictions that must change to alter the ensemble's final decision. This concept is analogous to the vote margin in classical voting theory.

For a majority vote ensemble of $n > 0$ models, voting robustness is simply the number of votes by which the leading class exceeds the runner-up (e.g., if 6 out of 10 models vote for the predicted class, at least two votes must flip to change the outcome). In soft-voting ensembles, robustness relates to the *confidence margin* – the gap between the averaged probability assigned to the predicted class and the next highest class. A large margin indicates that more substantial changes in individual model outputs are needed to alter the final prediction.

Voting robustness is particularly crucial for safety-critical AI systems, as high robustness ensures that even if some ensemble members fail or behave incorrectly, the system's overall decision remains unaffected.

2.3. Robustness metrics

To capture an ensemble's tolerance to component failures, we summarize and define several complementary metrics beyond raw accuracy.

Ensemble Accuracy: The accuracy of the ensemble on a test set under normal conditions, serving as the baseline for comparison.

Accuracy Drop: The reduction in accuracy when one or more models are corrupted or removed. A robust ensemble should degrade gracefully, typically losing less than 1% accuracy when a single member is compromised.

Class Switching Probability (CSP): The probability that the ensemble's predicted class changes when a single model's prediction changes. Low CSP values mean that no single model has undue influence on the decision.

Ensemble Margin: The difference between the ensemble's support for the predicted class and the runner-up class. A larger margin indicates greater decision stability, meaning that individual model failures are less likely to overturn the ensemble's output.

Leave-One-Out (LOO) Impact: A diagnostic analysis where each model is removed in turn to measure the effect on predictions and accuracy. Significant accuracy drops or frequent class changes upon removal reveal that a model is critical to the ensemble's decision-making.

2.4. Formal verification of neural networks

The formal methods community has developed several approaches for verifying properties of neural networks, including Satisfiability Modulo Theories (SMT) solving, linear programming (LP), and abstract interpretation. These methods have been applied to tasks such as verifying robustness to input perturbations and ensuring other safety properties [8, 11].

Tools such as Marabou [8, 13] encode a network's ReLU activation constraints as LP constraints, enabling solvers to determine whether specific outputs can be altered under given conditions. These techniques provide *certified guarantees* (e.g., proving that no adversarial example exists within a bounded input region).

However, most verification research to date has focused on single networks [2, 3, 7]. Ensemble models introduce additional complexity, particularly when using discrete majority voting, which creates combinatorial branching. Encoding majority voting in a formal verification setting is challenging due to the large number of possible vote distributions, significantly increasing the combinatorial complexity.

By contrast, averaging (soft voting) produces continuous outputs that can be expressed as an additional network layer, effectively *fusing* the ensemble into a single verifiable network. This transformation enables existing verification pipelines – whether based on SMT, LP, or abstract interpretation – to be applied to ensemble models without incurring the combinatorial explosion caused by discrete voting schemes.

2.5. Related work

Adversarial training with generated examples has been explored to close the robustness gap, and ensemble adversarial training has demonstrated that ensemble diversity improves defense effectiveness [4].

Formal verification of neural networks is an active field, with recent advances extending verification techniques to a wider class of activation functions and input sets. For example, Antal et al. [1] generalize verification methods for piece-wise linear activation functions, supporting both bounded and unbounded input domains, and demonstrate their effectiveness on multiple case studies.

Our work builds directly on these insights by adopting soft voting for ensemble verification, thereby enabling formal analysis while preserving robustness. To our knowledge, this is among the first studies to *verify ensemble voting robustness formally*, combining empirical results with SMT-based verification to certify that ensemble predictions remain stable even when some members are corrupted or removed.

3. Methodology

We designed an experiment to empirically evaluate the robustness of ensemble voting across multiple neural network architectures and simulated model failures. Our

methodology encompasses the neural network models employed, the construction of ensembles, the introduction of failures via corrupted models, and the evaluation procedure. We focus on classification of 10-digit classes (0–9), a scenario where some models might be "blind" to certain digits to simulate partial failure.

3.1. Neural network models

We consider five types of neural network (NN) architectures of varying complexity:

SimpleLinear: A simple linear classifier (logistic regression) with no hidden layer, directly mapping 784 input features to 10 class scores. This model has 7,850 parameters (784×10 weights + 10 biases) and serves as a minimal baseline.

HiddenMLP: A multi-layer perceptron (MLP) with one hidden layer of 64 units and ReLu activation, adding non-linearity and capacity compared to SimpleLinear. The model maps 784 inputs to 64 hidden units and then to 10 class scores through the hidden layer for a total of 50,370 parameters.

SingleConv: A small convolutional neural network (CNN) with a single convolutional layer of 32 filters of size 3×3, followed by flattening and a dense layer of 10 units. This model has approximately 3,074 parameters. It enables spatial feature extraction while keeping the network lightweight.

TinyCNN: A slightly deeper CNN with a single convolutional layer of 16 filters of size 3×3 and a dropout layer, followed by flattening and a dense layer of 10 units. This model has 1,498 parameters and achieves higher accuracy on digit classification due to its additional depth.

CompressedModel: A compressed MLP with a single hidden layer of 32 units and ReLU activation, mapping 784 inputs to 32 hidden units and then to 10 class scores through the hidden layer for a total of 25,370 parameters. This simulates scenarios where model size is constrained, potentially at some cost to accuracy.

Each architecture was trained on the digit classification task. We trained 17 *full* models per architecture, where "full" means the model was trained on the complete set of 10 digit classes (0–9). These 17 models were independently trained with different random initializations to provide diversity.

In addition to full models, we trained *corrupted* models to simulate omission failures. For each architecture, we trained models on datasets where 3 out of the 10 digit classes were omitted from the training set (the model never saw those classes and would likely misclassify them). We created four corruption schemes:

- **Scheme 0:** Models trained without digits $\{0, 1, 2\}$.
- **Scheme 1:** Models trained without digits $\{1, 2, 3\}$.
- **Scheme 2:** Models trained without digits $\{2, 3, 4\}$.
- **Scheme 3:** Models trained without digits $\{3, 4, 5\}$.

For each scheme, we trained seven models (28 corrupted models per architecture). The multiple models per scheme account for randomness in training and reduce bias from any single corrupted instance.

3.2. Ensemble construction

From the pool of 17 full models for each architecture, we constructed 100 ensembles, each consisting of 5 models. Each ensemble was created by randomly selecting 5 models out of the 17, ensuring diverse combinations.

While it is standard practice to balance ensembles by avoiding concentration of the best or worst models in the same group, in our case the performance deviations between models were so small that we omitted this step.

Each ensemble used *soft voting*: each model produced a probability distribution over the 10 classes (via softmax), and the probabilities were averaged element-wise. The predicted class was the one with the highest average probability. Soft voting naturally allows confident models to influence the decision while outliers are diluted by the consensus.

3.3. Simulating model failure (corruption schemes)

To evaluate robustness, we simulated single-model failures by replacing one model in an ensemble with a corrupted version of that model. For each of the 100 original ensembles, we generated four corrupted ensembles, one for each corruption scheme defined in Section 3.1.

Precisely one of the 5 ensemble members was swapped out, representing a single-point failure. The first model in each ensemble was replaced with a corrupted model from the same architecture's corrupted model pool. This yielded 100 corrupted ensembles per scheme, per architecture. All ensembles (both original and corrupted) employed the same soft voting mechanism.

3.4. Evaluation procedure

We evaluated every ensemble (and each corrupted variant) on a common test dataset of digit images specifically the MNIST dataset[9]. For each ensemble, we computed the metrics defined in Section 2: *Ensemble Accuracy*, *Ensemble Margin*, *Accuracy Drop*, and *Class Switching Probability (CSP)*. We also conducted a leave-one-out (LOO) analysis, in which each model was removed in turn to assess its impact on predictions.

The entire experimental pipeline – model training, ensemble construction, corruption injection, and evaluation – was implemented using TensorFlow/Keras. Trained models were saved and converted to the ONNX (Open Neural Network Exchange) format for interoperability with verification tools. ONNX conversion also enabled formal analysis using solvers by representing the ensemble as a single verifiable network with a soft-voting layer.

All metrics were computed from model predictions using Python scripts. We relied on `numpy` for statistical aggregation (e.g., averaging ensemble accuracies across 100 random ensembles per architecture and calculating standard deviations). When comparing original vs. corrupted predictions, we ensured the exact same test inputs were used and fixed all sources of randomness so that predictions were deterministic and reproducible.

Automated scripts generated random 5-model ensemble compositions from the pool of trained models and systematically injected corrupted models for each corruption scheme. Each original ensemble produced four corrupted versions (one per scheme), ensuring fair and uniform comparisons across all architectures.

All accuracy measurements, CSP calculations, and leave-one-out statistics were stored in structured logs and tables for traceability. This structured workflow enables the empirical results to align directly with the models used for formal verification, ensuring consistency between the experimentation and verification stages.

3.5. Formal verification on voting robustness

In order to encode NN models as a set of mathematical constraints, we rely on Marabou [8, 13]. The tool provides functionality for loading NN models from ONNX files and encoding them as a combination of linear equalities and inequalities, piecewise-linear constraints, and disjunctive constraints. Since Marabou does not support the encoding of Softmax layers due to their non-linearity, all Softmax layers must be removed in advance. Instead, we use the raw output values (logits) directly and encode the Argmax operation by introducing disjunctive constraints over the logits.

When loading multiple NN models and merging them into a single Marabou network, variable indexing must be handled carefully: (1) the input variables must be shared among all NNs, and (2) all other variables, including the outputs, must be *shifted* by an offset, computed on the fly, to avoid collisions in variable indices.

Encoding averaging for soft voting

To implement a soft voting scheme, the average of the output logits for each class must be computed. In Marabou, averaging can be encoded simply by summing the output variables and equating the result to a fresh variable – division by the number of models is unnecessary for verification purposes.

Let $n > 0$ be the number of NN models in the ensemble, and $m > 0$ the number of output classes. Let the variables o_{ij} denote the jth output logit of the ith NN model, where $1 \leq i \leq n$ and $1 \leq j \leq m$. We define s_j as a *fresh variable* representing the sum of logits for class j across all models:

$$\forall j \in \{1, \ldots, m\} : \quad \sum_{i=1}^{n} o_{ij} = s_j.$$

Encoding distinct argmax outputs

Let $s_1, \ldots, s_m$ and $s'_1, \ldots, s'_m$ denote the summed logits for two different ensembles (or models). We wish to encode the condition that the two ensembles predict *different* classes after applying Argmax. This can be expressed as the following disjunction:

$$\bigvee_{i,j=1}^{n} \left(\left(\bigwedge_{k=1}^{n} (s_i \geq s_k) \right) \wedge s'_i < s'_j \right).$$

Since Marabou supports only non-strict inequalities, the strict inequality $s'_i < s'_j$ must be rewritten as $s'_i \leq s'_j - \epsilon$, where $\epsilon > 0$ is a tunable constant controlling the precision for non-strict inequalities.

4. Experimental results

4.1. Experimental environment

All experiments were conducted on a cluster of 20 identical machines, forming a distributed computing setup. Each machine was equipped with an 8-core Intel(R) Xeon(R) W-2225 CPU at 4.10 GHz, 32 GB of RAM, and an NVIDIA RTX A4000 with 16 GB of VRAM. This setup allowed experiments to be parallelized across multiple machines. While this hardware configuration provided adequate computational resources for the neural network training phase and verification of simpler architectures, memory limitations became apparent during formal verification of more complex models.

The software environment consisted of TensorFlow 2.15 for training individual neural network models, NumPy for ensemble voting and statistical aggregation, Marabou 2.0 for formal verification encoding and constraint solving, and Gurobi 1.2.3 as the underlying optimization solver. The distributed setup enabled parallel execution of the 100 ensemble trials across multiple architectures and corruption schemes, significantly reducing the overall experimental runtime.

Following the training phase described in Section 3, all trained models were saved and subsequently converted to the ONNX (Open Neural Network Exchange) format to ensure compatibility with the Marabou verification framework. This conversion process maintained the mathematical equivalence of the models while enabling the constraint-based encoding required for SMT-based verification.

4.2. Empirical results

Now we present the empirical results demonstrating the impact of single-model corruption on ensemble performance over 100 random trials. Table 1 presents ensemble evaluation metrics for the different neural network architectures under both non-corrupted and corrupted conditions across four corruption schemes (S0, S1, S2, S3). The **Model** column represents the different neural network architectures. The **Corr.** column corresponds to the different corruption schemes and

their absence. The label "none" references the scenario with ensembles composed entirely of fully trained models, serving as a reference baseline, while S0...S3 correspond to ensembles where one model has been replaced by a corrupted model according to the specified corruption scheme. **Acc.** shows the average prediction accuracy over the trials. **Acc. Drop** indicates the decrease in accuracy relative to the baseline. **Margin** represents the average difference between the top two predicted class probabilities, quantifying prediction confidence. **CSP** (Class Switching Probability) measures how often the ensemble prediction changes when one member is removed or corrupted. **LOO Drop** quantifies the impact of leaving out a single model on ensemble accuracy.

4.2.1. Accuracy

As shown in the results, the accuracy did not vary significantly even when one model in the ensemble was replaced by a corrupted one. The largest observed accuracy drop is only 0.16% (see SimpleLinear in Table 1), demonstrating the robustness of our ensemble approach against individual model failures. This robustness is critical in maintaining reliable performance despite the presence of corrupted inputs.

4.2.2. Margin

While accuracy remained relatively stable, the ensemble margin consistently decreased under corruption, with the largest reduction being approximately 12%. Even the best performing model TinyCNN experienced a margin drop of roughly 11-12%. Nonetheless, the ensemble voting mechanism ensures that the corrupted model could not outweigh the consensus of the remaining four models, preserving decision stability.

4.2.3. Class Switching Probability (CSP)

The ratio of class switches, measured by the CSP, remained quite small; this indicates that our ensembles are stable under perturbations caused by corrupted models. CSP quantifies the proportion of samples for which an individual model alone changes the voting outcome. A low CSP implies that the corrupted model rarely influences the ensemble's decision, enabling rapid SAT verification by the Marabou tool. High CSP values correlate with easier SAT input detection, while low CSP values correspond to more robust ensembles that resist corrupted influences.

4.2.4. Leave-One-Out (LOO) drop

The LOO drop, which measures the accuracy decrease when removing one model from the ensemble, reflects the accuracy drop seen with corrupted models. Smaller LOO drops indicate that even reduced ensembles perform comparably to the full ensemble, emphasizing the resilience of the voting mechanism. A large LOO drop signifies a model's key role in correct decisions, meaning if corrupted, the ensemble is more likely to encounter SAT conditions. Conversely, small LOO drops suggest

Table 1. Evaluation results for different ensemble architectures (non-corrupted and corrupted).

Model	Corr.	Acc.	Acc. Drop	Margin	CSP	LOO Drop
SimpleLinear	–	0.92570	0.00000	0.84442	0.00000	0.00000
	S0	0.92430	0.00140	0.74527	0.00960	-0.00140
	S1	0.92410	0.00160	0.75020	0.01130	-0.00160
	S2	0.92450	0.00120	0.75910	0.01210	-0.00120
	S3	0.92460	0.00110	0.76557	0.00990	-0.00110
HiddenMLP	–	0.97430	0.00000	0.95354	0.00000	0.00000
	S0	0.97480	-0.00050	0.84248	0.00388	0.00050
	S1	0.97490	-0.00060	0.84143	0.00310	0.00060
	S2	0.97350	0.00080	0.84680	0.00380	-0.00080
	S3	0.97390	0.00040	0.85247	0.00420	-0.00040
SingleConv	–	0.98148	0.00000	0.97727	0.00005	-0.00001
	S0	0.98162	-0.00013	0.86241	0.00291	0.00013
	S1	0.98216	-0.00067	0.86128	0.00263	0.00067
	S2	0.98211	-0.00062	0.86461	0.00272	0.00062
	S3	0.98212	-0.00063	0.87166	0.00311	0.00063
TinyCNN	–	0.98232	0.00000	0.96820	0.00005	0.00000
	S0	0.98345	-0.00113	0.85642	0.00296	0.00113
	S1	0.98319	-0.00088	0.85450	0.00304	0.00088
	S2	0.98230	0.00002	0.85936	0.00332	-0.00002
	S3	0.98239	-0.00008	0.86576	0.00308	0.00008
CompressedM.	–	0.96500	0.00000	0.93244	0.00000	0.00000
	S0	0.96649	-0.00149	0.82676	0.00649	0.00149
	S1	0.96550	-0.00050	0.82351	0.00550	0.00050
	S2	0.96430	0.00070	0.83104	0.00570	-0.00070
	S3	0.96420	0.00080	0.83674	0.00560	-0.00080

that corrupted models have limited impact on voting outcomes, often resulting in UNSAT conditions. In our case, the LOO Drop is equal to the inverse of the accuracy drops of the corrupted ensembles, showing us that the smaller ensemble containing 4 full models is just as strong as our full 5 model ensemble.

Notable observations In some cases, ensembles containing a corrupted model appear to achieve slightly higher accuracy than the corresponding full ensemble. This behavior is likely due to the combination of two factors: (i) the MNIST dataset is relatively small and well-separated, so omitting a few classes in one model does not drastically affect predictions on the test set, and (ii) the corruption itself is mild, meaning the corrupted model may occasionally make correct predictions by chance that complement the other ensemble members. Consequently, random variations across the 100 ensemble trials can produce instances where the corrupted ensemble performs marginally better than the full ensemble. Importantly, these occurrences are rare and do not undermine the overall robustness trends observed across all metrics.

4.3. Formal verification results

We now present the verification results for the ensemble models described above. In our experiments, we used the Marabou solver to assess whether it could complete verification tasks within a timeout of 1200 seconds, and – most importantly – whether it could return a correct SAT (satisfiable) or UNSAT (unsatisfiable) result for different NN models under various corruption schemes.

For each NN model, we constructed 50 ensemble instances by randomly selecting 5 full models and 1 corrupted model. We then executed three distinct verification tasks on each ensemble:

Model Failure: Checks whether the full ensemble and a corrupted ensemble (in which the first intact model is replaced by the corrupted one) can produce different class predictions. If the verification result is SAT, then such inputs exist; if it is UNSAT, the two ensembles always agree on their prediction. In SAT cases, our tool also returns a set of input values as a counterexample.

Leave-One-Out (LOO): Checks whether the corrupted ensemble and the ensemble from which the corrupted model is entirely removed can predict different classes. The SAT/UNSAT interpretation is the same as in the Model Failure task.

Ensemble Equivalence: Checks whether two instances of the corrupted ensemble can produce different predictions. This task was implemented primarily to study how the solver handles UNSAT instances, as verifying UNSAT cases is generally considered more computationally demanding than verifying SAT cases.

Table 2 provides insights into the scale of the verification problem instances solved for different types of ensemble models and verification tasks. The column **#Vars** denotes the number of decision variables, while **#ReLUs** represents the number of ReLU constraints. The **#Eqs** column reports the total number of equations and inequalities, followed by two statistical measures describing the number of addends in these constraints – **Mean** indicates the average number of addends, and **Median** shows the corresponding median value.

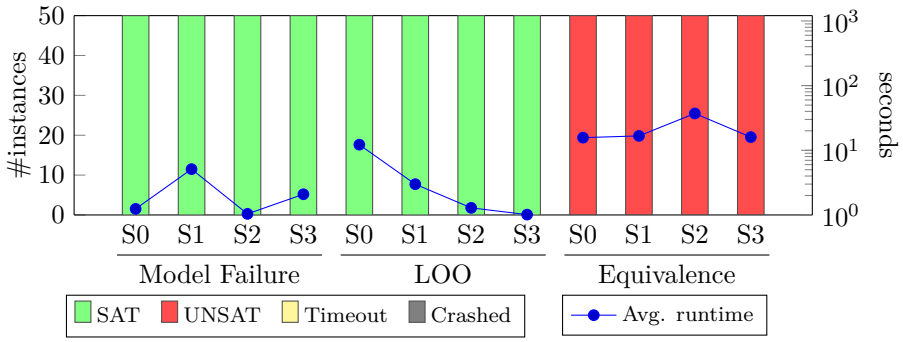
Figures 1, 2, and 3 present the verification results for the SimpleLinear, HiddenMLP, and CompressedModel ensembles, respectively. The bars are grouped according to the three verification tasks, with each group containing results for the four corruption schemes (S0, S1, S2, S3). The stacked bars display the distribution of SAT and UNSAT results, timeouts, and solver crashes across the 50 ensemble instances. The line plot shows the solver's average runtime in seconds (note that the right-hand vertical axis is log-scaled).

As expected, most of the Model Failure and LOO instances are SAT, indicating that the ensembles are not completely robust with respect to their voting behavior. For these SAT cases, the solver successfully found counterexamples (adversarial inputs), even though these inputs were artificially crafted by the solver and are not part of the original dataset. However, some Model Failure and LOO instances were proven to be UNSAT, meaning that they are *absolutely robust* – the solver

Table 2. Statistics on the verification problem instances as modeled by Marabou.

Model	Ver. Task	#Vars	#ReLUs	#Eqs	Mean	Median
SimpleLinear	Model Failure	864	0	80	590	785
	LOO	854	0	70	562	785
	Equivalence	844	0	60	655	785
HiddenMLP	Model Failure	1632	384	464	658	785
	LOO	1494	320	390	653	785
	Equivalence	1484	320	380	670	785
SingleConv	Model Failure	260448	129792	129872	20	10
	LOO	217174	108160	108230	20	10
	Equivalence	217164	108160	108220	20	10
TinyCNN	Model Failure	130656	64896	64976	20	10
	LOO	109014	54080	54150	20	10
	Equivalence	109004	54080	54140	20	10
CompressedM.	Model Failure	1248	192	272	562	785
	LOO	1174	160	230	554	785
	Equivalence	1164	160	220	579	785

Figure 1. Verification results for **SimpleLinear** ensembles.



was able to guarantee that no adversarial inputs exist.

Interestingly, for all SingleConv and TinyCNN ensembles, the solver crashed in every case due to the system running out of available RAM. We suspect that the solver was unable to handle the computational overhead introduced by convolutional layers in these models, though this issue requires further investigation. Similarly, we plan to investigate why the solver occasionally crashed when checking the equivalence of HiddenMLP and CompressedModel ensembles (see Figures 2 and 3).

We note that a few UNSAT instances were incorrectly reported as SAT by the solver. However, a manual review of the logs revealed that, in these cases, the two ensembles under investigation produced identical predictions. The apparent discrepancy arose from the encoding used for comparing Argmax outputs (see Sec-

Figure 2. Verification results for **HiddenMLP** ensembles.

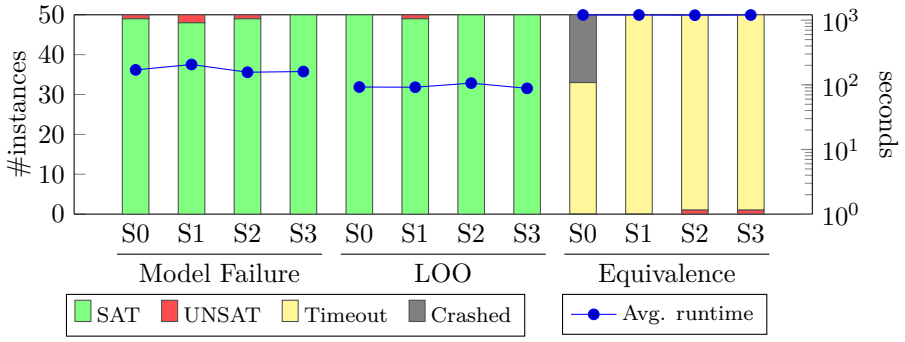
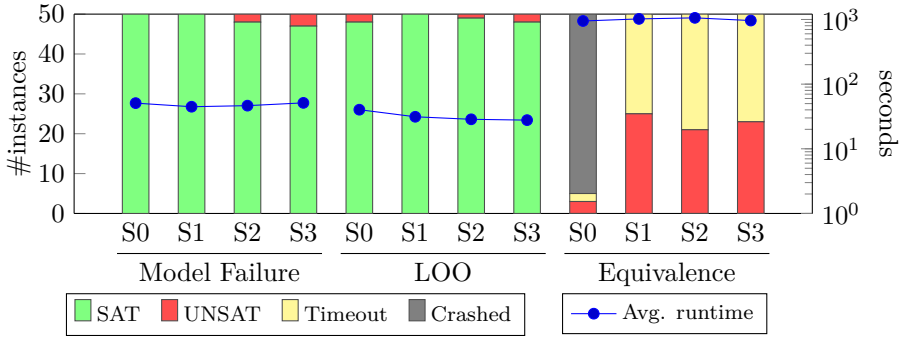


Figure 3. Verification results for **CompressedModel** ensembles.



tion 3.5), where the predicted values differed by less than the chosen precision threshold of $\epsilon = 0.001$.

5. Conclusion

Robustness in neural network ensembles is paramount for safety-critical AI applications, where high reliability is required even if some individual models fail. In this work, we introduce voting robustness as a measure of an ensemble's tolerance to model failures and demonstrate, through extensive experiments, that soft-voting ensembles can maintain accurate and stable predictions despite single-model corruption. Empirically, our results showed a negligible drop in accuracy (less than 1% in the worst case) when one network was compromised, confirming that the ensemble's performance degrades gracefully. No single member has undue influence on the final decision, and the consensus of the ensemble preserves overall reliability beyond what any standalone model could achieve.

Equally important, we combined empirical analysis with formal verification

to provide a rigorous assurance of robustness. Using an SMT-based verifier, we encoded the entire soft-voting ensemble into a single model and demonstrated that, in many cases, the ensemble's prediction remains unchanged under any single-model failure. The formal analysis not only confirmed the ensemble's resilience observed in testing but also identified corner-case adversarial inputs for certain ensembles, highlighting that empirical robustness alone does not guarantee absolute security. By uniting these approaches, our study offers both practical evidence and provable guarantees of robustness. This comprehensive evaluation enhances confidence in soft-voting ensembles and demonstrates the value of integrating empirical results with formal methods to verify the reliability of AI systems.

The challenges we encountered with current verification engines (e.g., solver memory exhaustion on convolutional networks and occasional precision-related errors) point to the need for more scalable and reliable neural network verifiers. Future verification frameworks should enhance their handling of complex architectures and large ensembles – for instance, by optimizing memory usage or incorporating problem-specific heuristics – so that formal certification can keep pace with the growing complexity of models. Furthermore, maximizing the vote margin and limiting the influence of any single model will not only improve empirical resilience but also facilitate formal verification.

Our study focused on single-model failure scenarios in a controlled classification task, and the SMT-based verification struggled with very deep or convolutional models due to scalability constraints. These limitations mark directions for future work. Exploring multiple simultaneous model failures, applying our framework to larger real-world datasets, and devising more efficient verification algorithms are natural next steps to generalize our approach. Despite these challenges, our work demonstrates a promising step toward neural network ensembles that combine high accuracy with provable robustness.

Code availability

The trained ensemble models, their ONNX representations, and all scripts used in the experiments are openly accessible at our Zenodo repository: https://doi.org/10.5281/zenodo.17286858.

References

[1] L. Antal, E. Abraham, H. Masara: *Generalizing neural network verification to the family of piece-wise linear activation functions*, Science of Computer Programming 243 (2025), p. 103269.

[2] R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, P. K. Mudigonda: *A Unified View of Piecewise Linear Neural Network Verification*, in: Advances in Neural Information Processing Systems (NeurIPS), Curran Associates, Inc., 2018, pp. 4795–4804, doi: 10.48550/arXiv.1711.00455.

[3] R. Ehlers: *Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks*, in: Automated Technology for Verification and Analysis (ATVA), vol. 10482, Lecture Notes in Computer Science, Springer, 2017, pp. 269–286, doi: 10.1007/978-3-319-68167-2_19.

[4] S. Gowal, C. K. Qin, J. Uesato, T. Mann, P. Kohli: *Improving Robustness using Generated Data*, Advances in Neural Information Processing Systems 34 (2021), pp. 4211–4224, doi: 10.48550/arXiv.2110.09468.

[5] D. Gross, N. Jansen, G. A. Pérez, S. Raaijmakers: *Robustness Verification for Classifier Ensembles*, in: Automated Technology for Verification and Analysis, 2020, pp. 271–287, doi: 10.1007/978-3-030-59152-6_15.

[6] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, D. Song: *Natural adversarial examples*, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021, pp. 15262–15271.

[7] G. Katz, C. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer: *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*, in: Computer Aided Verification (CAV), vol. 10426, Lecture Notes in Computer Science, Springer, 2017, pp. 97–117, doi: 10.1007/978-3-319-63387-9_5.

[8] G. Katz, C. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer: *The Marabou Framework for Verification and Analysis of Deep Neural Networks*, in: Computer Aided Verification (CAV), Springer, 2019, pp. 443–452, doi: 10.1007/978-3-030-25540-4_26.

[9] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner: *Gradient-based learning applied to document recognition*, Proceedings of the IEEE 86.11 (1998), pp. 2278–2324.

[10] T. Pang, K. Xu, C. Du, N. Chen, J. Zhu: *Improving adversarial robustness via promoting ensemble diversity*, in: International Conference on Machine Learning, PMLR, 2019, pp. 4970–4979.

[11] G. Singh, T. Gehr, M. Mirman, M. Püschel, M. Vechev: *An Abstract Domain for Certifying Neural Networks*, in: Proceedings of the ACM on Programming Languages (POPL), vol. 3, 2019, pp. 1–30, doi: 10.1145/3290354.

[12] F. Tramer, N. Carlini, W. Brendel, A. Madry: *On adaptive attacks to adversarial example defenses*, Advances in neural information processing systems 33 (2020), pp. 1633–1645.

[13] H. Wu, A. Ozdemir, A. Zeljić, K. Julian, A. Irfan, D. Gopinath, S. Fouladi, G. Katz, C. Pasareanu, C. Barrett: *Parallelization Techniques for Verifying Neural Networks*, in: Formal Methods in Computer Aided Design (FMCAD), IEEE, 2020, pp. 128–137, doi: 10.34727/2020/isbn.978-3-85448-042-6_20.