AIREPAIR: A Repair Platform for Neural Networks

Xidan Song*, Youcheng Sun*, Mustafa A. Mustafa*†, Lucas C. Cordeiro*‡
*Department of Computer Science, University of Manchester, UK
†imec-COSIC, KU Leuven, Belgium
‡Federal University of Amazonas, Brazil
*{xidan.song, youcheng.sun, mustafa.mustafa, lucas.cordeiro}@manchester.ac.uk

***Abstract*—We present AIREPAIR, a platform for repairing neural networks. It features the integration of existing network repair tools. Based on AIREPAIR, one can run different repair methods on the same model, thus enabling the fair comparison of different repair techniques. In this paper, we evaluate AIREPAIR with five recent repair methods on popular deep-learning datasets and models. Our evaluation confirms the utility of AIREPAIR, by comparing and analyzing the results from different repair techniques. A demonstration is available at https://youtu.be/UkKw5neeWhw.**

I. INTRODUCTION

As neural networks are widely applied in various areas, insufficient accuracy, adversarial attack, and data poisoning, among others, threaten the development and safe deployment of neural network applications [1]. Unlike traditional software, a neural network model is not programmed manually by software developers. Instead, its parameters are automatically learned from training data. As a result, these models' defects cannot be easily fixed.

Many repair techniques and tools, like DeepRepair [2], DL2 [3], Apricot [4], DeepState [5] and RNNRepair [6] have been proposed to automatically fix the defects found in a neural network Naturally, the neural network repair has two objectives: correcting the model's wrong behaviors on failed tests while not comprising its original performance on passing tests. Existing repair techniques can be classified into three categories: retraining/fine-tuning, direct weight modification, and architecture extension.

In the first category of repair methods, the idea is to retrain or fine-tune the model for the corrected output with the identified misclassified input. Typical methods include counterexample-guided data augmentation, which adds misclassified examples iteratively to the training datasets [7]. The editable training in [8] aims to efficiently patch a mistake of the model on a particular sample, and the training input selection in [9] emphasizes selecting high-quality training samples for fixing model defects. DeepRepair [2] implements transfer-based data augmentation to enlarge the training dataset before fine-tuning the models. RNNRepair [6] and DeepState [5] also augment the datasets and perform retraining to improve the performance of the neural network models; they are specifically designed to repair the Reconcurrent Neural Network (RNN) [10] models.

The methods in the second category calculate new weight values of the model to fix erroneous nodes in the neural network, to improve the classification accuracy, or to satisfy safety properties. In [11] and [12], SMT solvers are used for solving the weight modification needed at the output layer for the neural network to meet specific requirements without any retraining. In [13], the repair method localizes the faulty neuron weights and optimizes the localized weights to correct the model misbehavior by Particle Swarm Optimisation algorithm. The Apricot tool [4] adjusts the weights of a model with the feedback from a set of so-called reduced models trained with subsets of training data.

Repair techniques in the third category extend a given neural network architecture, e.g., by introducing more weight parameters or repair units, to facilitate more efficient repair. PRDNN [14] introduces Decoupled DNNs, a new DNN architecture that enables efficient and effective repair. DeepCorrect [15] corrects the worst distortion-affected filter activations by appending correction units. DL2 [3] implements neural networks that can train with constraints that restrict inputs outside the training set. Researchers have used these different methods to develop many repair tools. Different repair tools run on different environments with customized configurations. They accept different input formats and produce various outputs. It can be challenging to figure out which repair tool and setting to use. To make it easier for users to configure appropriate repair methods and parameters, we developed a neural network repair platform – AIREPAIR. It can automatically repair trained models using different methods. We believe that AIREPAIR would be helpful for developers who want to improve their neural networks and those interested in evaluating their new repair methods.

The main contributions of this paper are three-fold:

- We develop AIREPAIR for integrating and evaluating existing (and future) repair techniques on neural networks.
- We benchmark five repair techniques on 11 types of neural network models across four datasets.
- We make AIREPAIR and its benchmark publicly available: https://github.com/theyoucheng/AIRepair

II. AIREPAIR DESCRIPTION

A. Overview

The motivation for AIREPAIR is to develop a platform for testing and evaluating different repair methods in a compatible way. As illustrated in Fig. 1, AIREPAIR accepts the trained models and the training datasets if they are specified in the

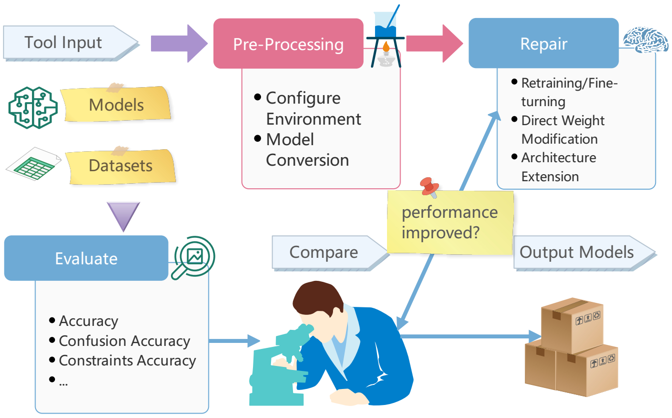

Fig. 1: The AIREPAIR Architecture.

configuration. It performs pre-processing on different benchmarks to make them capable of different frameworks. Pre-processing isolates different running environments for various deep learning libraries, e.g., TensorFlow [16] or PyTorch [17]. After the repair, AIREPAIR collects the results and analyses them automatically, which is done by examining the outputs and experimental logs. Finally, it presents the results for the user to decide which repairing tool suits their models. The output from AIREPAIR includes the repaired model combined with the logs and parameters.

B. Modular View

In this part, we detail each component in AIREPAIR: 1) input, 2) pre-processing, 3) repair, and 4) evaluation.

1) Input: The input to our platform is trained neural network models and testing or training datasets depending on the repair method configured. AIREPAIR accepts fully connected feed-forward neural networks and convolutional neural networks in popular deep learning model formats such as .pth and .pt from PyTorch, and .pb, and .h5 from TensorFlow + Keras [18]. Our platform has been tested on standard datasets like MNIST, Fashion-MNIST, CIFAR-10 and CIFAR-100.

2) Pre-processing: This component converts neural network models between different formats, configures the running environments for various underlying repair tools, and evaluates the models before repair.

3) Repair: Based on the input and pre-processing, the repair component configures and calls the underlying repair methods to perform repairs on the input models. During repair, it monitors the hardware resource consumption, displays messages for the status of the repair procedure, and saves the logs for evaluation.

4) Evaluation: This component measures the performance of a model before and after repair. There are several metrics for characterizing a model's performance, from complementary perspectives, including the model's *(classification) accuracy*, *constraint accuracy* [3], and *confusion accuracy*. The constraint accuracy describes the percentage of predictions given by the model that satisfies the constraint associated with the problem, which requires that the probabilities of groups of classes have either a very high or a very low probability. The confusion accuracy is defined as $P = \frac{TP}{TP+FP}$, where TP and FP are True Positive and False Positive classifications. These two metrics evaluate the model's robustness. Data augmentation of different kinds of blurs (glass, motion, and zoom) [19] can be applied on the input dataset when collecting these metrics.

Typically, existing repair tools take some misclassified inputs, and the repair goal is to correct those erroneous nodes. Each repair method often focuses on improving the performance according to one type of evaluation metric. The AIREPAIR tool integrates different repair methods and different performance metrics, to give a comprehensive view when repairing a neural network model.

5) Output: AIREPAIR delivers the repaired models after repair and the resulting configurations for each repair method that will be stored in log files.

C. Example Usage

At the first step, we encourage one to train the baseline model using the script ('train_baseline.py') provided. Subsequently, one can configure and run different network repair tools with AIREPAIR:

```
python AIRepair.py [-h] [--all]
[--net_arch NETARCH] [--dataset DATASET]
[--pretrained PATH_AND_FILENAME]
[--depth DEPTH]
[--method METHOD] [--auto]
[--additional_param PARAM]
[--input_logs INPUT_LOGS]
[--testonly]
```

For example, the setup below configures and runs AIREPAIR with three repair methods, Apricot, DeepRepair, and DL2, as discussed in Section I. They are applied to repair a model named 'cifar10_resnet34' with the CIFAR-10 dataset.

```
python AIRepair.py --method apricot
    deeprepair dl2 --pretrained
    cifar10_resnet34_baseline.pt --dataset
    cifar10 --net_arch resnet --depth 34
```

In particular, when using '--pre-trained' to specify the path of a trained neural network model to repair, users need to specify the model's architecture as well as depth. PyTorch has two different methods to save the trained model: saving the entire model or saving the state_dict or checkpoint. When loading the neural network model, AIREPAIR needs to know its structure for the second method. It has the built-in structure definition for ResNet families and several convolutional neural networks for MNIST and Fashion-MNIST. Hence, users only need to specify the net architecture and depth when loading the state_dict. For the architectures that do not belong to these three models, users either provide the entire model or customize AIREPAIR's pre-processing module. Currently, AIREPAIR can process both feed-forward neural networks and convolutional neural networks.

The parameter '--net_arch' specifies the architecture of models, and '--dataset' selects the corresponding dataset (that are

TABLE I: AIRepair results: The best accuracy (Acc.) and constrains accuracy (Const.) improvement for each model is highlighted in $\boxed{\%}$ and $\boxed{\%}$ separately.

Datasets		CIFAR-10			CIFAR-100			MNIST	Fashion-MNIST
Models		ResNet18	ResNet34	ResNet50	ResNet18	ResNet34	ResNet50	MNIST	Fashion-MNIST
Baselines	Acc.	92.05%	91.34%	94.42%	46.84%	44.16%	47.36%	99.45%	92.20%
	Const.	90.51%	90.27%	90.66%	86.62%	85.95%	85.21%	99.96%	100%
Apricot	Acc.	-2.65%	-0.38%	-3.4%	+9.02%	+13.74%	+11.15%	+0.06%	+0.61%
DeepRepair	Acc.	+0.5%	-1.27%	-4.14%	+10.91%	+21.42%	+20.32%	+0.17%	+0.47%
	Const.	-9.46%	-8.82%	-12.77%	-37.62%	-34.95%	-29.71%	-0.43%	-4.10%
DL2	Acc.	-2.16%	+0.23%	-1.95%	+0.87%	+1.17%	-1.16%	+0.08%	+0.28%
	Const.	+9.3%	+9.61%	+5.4%	-0.49%	-0.89%	-0.4%	+2.55%	+6.27%

needed for retraining/refining or attaching correction units). These are as discussed in Section II-B1. The specific repair method (tool) can be specified using '--method', and '--auto' will automatically invoke the repair using the default parameters for the selected repair methods. For example, it sets the following configurations for DeepRepair method to repair ResNet34 trained on CIFAR-10 dataset.

```
--batch_size 128 --lr 0.1 --lam 0
--extra 128 --epoch 60 --beta 1.0
--cutmix_prob 0 --ratio 0.9
```

Alternatively, one can use '-–additional_param' to customize these parameters, which will substitute these default settings. The '--testonly' option is for evaluating a model without any repair procedure; users can use it before or after repair to check the model's performance. The evaluation metrics include accuracy, confusion, and constraint accuracy. For convenience, 'python AIRepair.py --all' runs all the repair methods on all available models [1] with default parameters automatically. Note that this option requires substantial computing power.

III. Experiment

A. Experimental Setup

We ran experiments on a machine with Ubuntu 18.04.6 LTS OS Intel(R) Xeon(R) Gold 5217 CPU @ 3.00GHz and two Nvidia Quadro RTX 6000 GPUs. Five repair methods, Apricot [4], DeepRepair [2], DL2 [3], DeepState [5], and RNNRepair [6], are chosen as baselines. They are applied to repair a benchmark of 11 neural network models (including ResNet models [20]) with four datasets MNIST [21], Fashion-MNIST [22], CIFAR-10 and CIFAR-100 [23]. Users can invoke it by specifying '--auto' as discussed in Section II-C. We repeat the same experiment three times to eliminate randomness in repair methods.

B. Results

Table I shows the complete AIREPAIR results of 3 repair methods on 8 neural networks (columns) from 4 datasets. We measured the performance by accuracy and constraint accuracy. The baseline represents each model's actual performance, and for each repair method, we report the absolute performance increment or decrement after repair. Apricot repairs a neural network by directly modifying its weights parameter values. DeepRepair and DL2 belong to the category of retraining and architecture extension, respectively, for repairing neural networks. The three baselines are selected to represent all 5 categories of neural network repair methods, as discussed in Section I.

For the models trained on CIFAR-10, we can see that DL2 brings the highest constrains accuracy (Const.) improvement, while this often comes with slight drops in the plain accuracy (Acc.). The adversarial robustness and the plain accuracy are two contradicting goals [24]. Besides, the performance of the CIFAR-10 models drops to different extents by Apricot and DeepRepair. Apricot is not designed to improve the constrains accuracy of models. We do not evaluate the constrains accuracy metrics on models repaired by Apricot.

At the same time, Apricot and DeepRepair have better improvements on CIFAR-100 neural network models. This indicates that when the original model's performance is relatively low, the Apricot and DeepRepair are more effective. Moreover, as the ResNet models become deeper, from 18 layers to 50 layers, DeepRepair can improve accuracy by sacrificing constraint accuracy. DL2 has no significant improvement in performance this time.

The original implementations for Apricot, DeepRepair, and DL2 do not support models trained on MNIST and Fashion-MNIST, so we created patches to integrate them into AIREPAIR for pre-processing and repairing these datasets and models. As shown in Table I (last two columns), all three repair methods result in noticeable performance improvements of different levels for MNIST and Fashion-MNIST models, even though the original networks' performance is already high (99.45% and 92.20% respectively).

In summary, there are some general observations from the experiments. There is no single best repair method on all benchmarks and on all evaluation metrics. Different repair methods seem to complement each other highly, and this would motivate future "combined repair" of neural network models, for which AIREPAIR can serve as the test bed. Mostly, as expected, less complex neural networks (MNIST/Fashion-MNIST in Table I) and lower performance models (CIFAR-100 examples in Table I) are easier to repair. We still regard

[1] https://zenodo.org/record/7627801#.Y-X6g3bP3tU

them as valuable observations, as they confirm that AIREPAIR is a valid platform for benchmarking different repair methods.

More neural network architectures: We also tested AIREPAIR on three RNN-based architectures. We apply DeepState and RNNRepair to repair LSTM, BLSTM, and GRU models trained on MNIST. The baseline model classification accuracy is 98.6%, 98.54%, and 98.93%. DeepState slightly improves the accuracy of these models by 0.05%, 0.19%, and 0.18%, whereas the model accuracy drops by 0.09%, 0.06%, and 0.33% when using RNNRepair.

C. Discussions

Based on AIREPAIR experiments, when encountering the performance anomaly of a neural network model, we suggest using neural network repair methods in the following order: 1) direct weight modification, 2) fine-tuning/retraining, and 3) attaching a new repair structure into the model.

For simpler neural networks like models trained on MNIST and Fashion-MNIST datasets, we put direct weight modification as a higher priority for repair. This repair does not require training/testing datasets and can be done without GPU hardware. However, it may face more significant challenges when repairing a trained model with high performance.

For complex convolutional models such as ResNets with millions of parameters, direct weight modification is more likely to encounter the search space explosion problem. In such cases, retraining/refining-based repair methods can still be applied without altering the neural network's structure. However, they may require more powerful hardware support.

If there is over-fitting during the retraining procedure or even further improvement is needed for the repair, or the models need to improve specific constraint accuracies, one could consider attaching repairing units.

Determining which repair method to use may depend on the NN application scenarios. For example, NN models trained on the MNIST dataset recognize handwriting materials, while models trained on ImageNet are designed to be used on image-based searches. We suggest referring to the above discussion and prioritizing the selection of the appropriate repair according to the characteristics and types of models. Another unique use case is the quantized neural network models used in mobile and embedded devices.

IV. CONCLUSION

We present AIREPAIR, a comprehensive platform for repairing neural networks, and it can test and compare different neural network repair methods. This paper gives the results of five existing neural network repair tools integrated into AIREPAIR. Although AIREPAIR is an early prototype, it shows promising results. We will support and test more neural network repair methods and propose a unified interface for developers to test and benchmark their repair methods.

ACKNOWLEDGEMENTS

This work is funded by the EPSRC grant EP/T026995/1 *"EnnCore: End-to-End Conceptual Guarding of Neural Architectures"*. The Dame Kathleen Ollerenshaw Fellowship of The University of Manchester supports M. A. Mustafa.

REFERENCES

[1] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[2] B. Yu, H. Qi, Q. Guo, F. Juefei-Xu, X. Xie, L. Ma, and J. Zhao, "Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment," *IEEE Transactions on Reliability*, 2021.

[3] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, and M. Vechev, "DL2: Training and querying neural networks with logic," in *ICML*, 2019.

[4] H. Zhang and W. Chan, "Apricot: A weight-adaptation approach to fixing deep learning models," in *ASE*. IEEE, 2019.

[5] Z. Liu, Y. Feng, Y. Yin, and Z. Chen, "DeepState: selecting test suites to enhance the robustness of recurrent neural networks," in *ICSE*, 2022.

[6] X. Xie, W. Guo, L. Ma, W. Le, J. Wang, L. Zhou, Y. Liu, and X. Xing, "Rnnrepair: Automatic rnn repair via model-based analysis," in *ICML*, 2021.

[7] T. Dreossi, S. Ghosh, X. Yue, K. Keutzer, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Counterexample-guided data augmentation," *arXiv:1805.06962*, 2018.

[8] A. Sinitsin, V. Plokhotnyuk, D. Pyrkin, S. Popov, and A. Babenko, "Editable neural networks," *arXiv:2004.00345*, 2020.

[9] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "MODE: automated neural network model debugging via state differential analysis and input selection," in *ESEC/FSE*, 2018.

[10] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[11] B. Goldberger, G. Katz, Y. Adi, and J. Keshet, "Minimal modifications of deep neural networks using verification." in *LPAR*, 2020, p. 23rd.

[12] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. S. Păsăreanu, "Nn repair: Constraint-based repair of neural network classifiers," in *CAV*. Springer, 2021, pp. 3–25.

[13] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search based repair of deep neural networks," *TOSEM*, 2022.

[14] M. Sotoudeh and A. V. Thakur, "Provable repair of deep neural networks," in *PLDI*, 2021.

[15] T. S. Borkar and L. J. Karam, "Deepcorrect: Correcting dnn models against image distortions," *IEEE Transactions on Image Processing*, vol. 28, no. 12, pp. 6022–6034, 2019.

[16] M. Abadi, P. Barham, Chen *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *USENIX OSDI*, 2016.

[17] A. Paszke, S. Gross, F. Massa *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NeurIPS*, 2019.

[18] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[19] A. Laugros, A. Caplier, and M. Ospici, "Are adversarial robustness and common perturbation robustness independant attributes?" in *ICCV*, 2019.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[21] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, 2012.

[22] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv:1708.07747*, 2017.

[23] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "There is no free lunch in adversarial robustness (but there are unexpected benefits)," *arXiv:1805.12152*, vol. 2, no. 3, 2018.