Lab 02: Machine Learning Attack on PUF

Jasmin Kaur and Alec Braynen
University of South Florida
Tampa, FL 33620

I. Introduction

Machine learning (ML) is a branch of Artificial Intelligence (AI) which is often used to automate the process of modeling data for data analysis by learning the patterns and behaviours of complex data [1]. It involves training mathematical models known as artificial neural networks (ANNs) using representation learning in order to model data behaviour [1]. ANNs are modeled as interconnected networks of artificial neurons where each neurons have one or more inputs along with a *weight* and a *bias*. The output of one such neuron is then passed through an *applied function* for the final output [1].

When several layers of artificial neurons exists between the input and output layers, they form a deep neural network (DNN). Usually, it requires a large data set and power to train a DNN. There are three steps involved while training a DNN - training, testing, and validation [1]. Training of an ML model can be fine tuned using *hyperparameters* such as batch size, number of epochs, learning rate as well as the model architecture to make the model predictions more accurate [1].

In this work, the authors experiment with modeling an Arbiter Physically Uncloneable Function (PUF), a 2 BIT XOR APUF and a 4 BIT XOR APUF using a neural network and a multi-layered perceptron (MLP) model. The algorithms are given challenge-response pairs as input for training. These challenge response pairs are generated from the challenge bits given to a PUF and its returned response bits. The algorithms are trained on this data with variety on the percentage splits of the dataset into training and testing data, i.e. (5% as training, 95% as testing, then 10% as training and 90% as testing and so on.) The results of this work show that in general, a larger training set of data leads to better accuracy on the modeling of a PUF, especially with the MLP model, however there are some discrepancies where less data can have give better accuracy with the DNN model.

II. Reading Check

Question 1: What makes neural networks good for modeling PUFs?

Answer: Neural networks' high ability to recognize and learn the patterns/features of a complex data set, i.e. the challenge and response bits of a PUF. Therefore, training a model on the challenge response pairs data set from a PUF allows it to "learn"/model the behaviour/functionality of a PUF and therefore predict the PUFs outputs. [1].

Question 2: Why is it important to split the data into separate training, validation and testing sets?

Answer: The training data set is used to train an ML model to learn and recognize the patterns of a complex data. The validation data set is used to validate the performance of an ML model during its training process. This allows one to adjust hyperparameters and other settings for better model performance. Finally, the testing data set is used to test the trained ML model on unseen data [1]. One main importance of splitting the data into separate training, validation, and testing sets is to avoid the issue of *overfitting*, where the trained ML model is able to make good predictions on sample/training data set but not on the unseen/testing data set. There needs to be unseen data in the training, validation and testing process to help the model learns generalized features.

Question 3: What features/data are used by the neural network to model the PUF?

Answer: For a given PUF, its given challenge data and its response data from the challenge is obtained and paired together to be used by the neural network as a feature/data source. More specifically, we use the parity vector feature of each challenge as input to the neural network model [1].

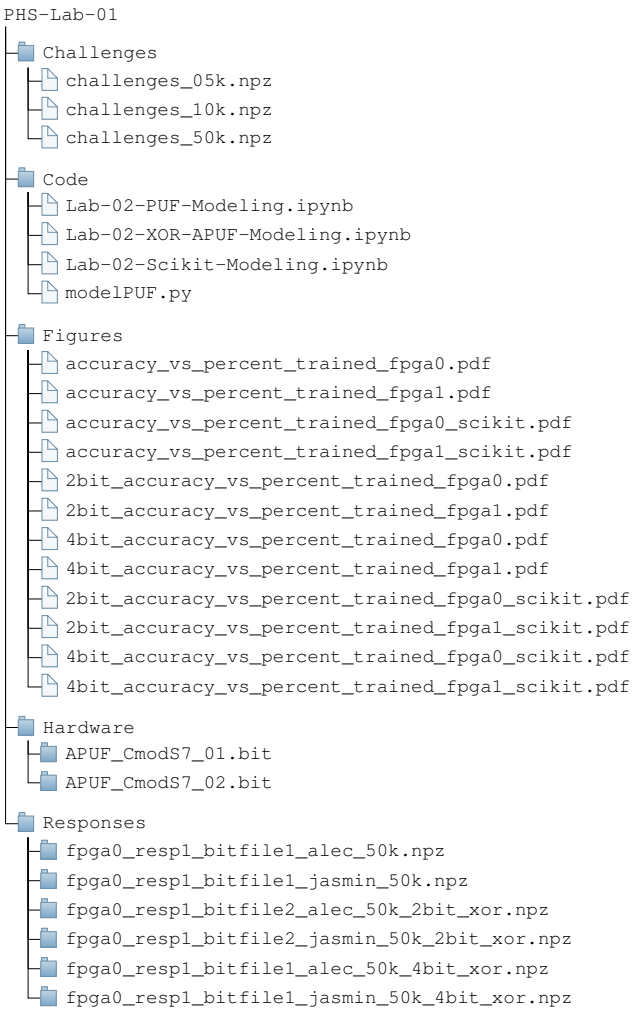
III. Methods

To implement this experiment, we use the 50,000 challenge and response bits of the two CMOD S7 FPGAs generated in a previous experiment, a virtual machine running Ubuntu OS, Jupyter Notebooks/Google collab, the programming language Python, and the libraries, NumPy, Matplotlib, TensorFlow v2.0, and SciKit Learn. The PUF neural network model and the data pre-processing code was supplied to us and codified in the file "modelPUF.py". Note, no physical FPGA hardware is interfaced with in this lab.

A. Software Setup

The software setup for the experiment consists of a virtual machine running Ubuntu OS on Virtualbox. The Python code is implemented via Jupyter notebooks on this virtual machine. Tensorflow v2.0 is used to interface the ML application used to create the model of a PUF.

The directory structure of the files in the Jupyter notebook is structured as follows:



1) The "Challenges" directory holds the challenge vectors used as inputs.
2) The "Code" directory hold the jupyter notebook files where the python code is executed.
3) The "Figures" directory holds the generated figures of the analyzed data.
4) The "Hardware" directory holds the APUF model bit files.
5) The "Response" directory holds the responses received from the FPGAs as output.

B. Experimental Procedure

Part A: Using Python script - "modelPUF.py"

The Python script "modelPUF.py" is used as follows:

- The "modelPUF.py" is used instantiate a neural network model of an APUF. Additionally, SciKitLearn's MultilayeredPerceptron class is used to instantiate a MLP model.
- Relevant Python libraries - NumPy, SciKitLearn, SciPy, and Tensorflow are imported.
- The ChallengeResponseSet class is used to parse and preprocess the challenge-response data into random train and test subsets based on the split percentage.
- The pufModel class is used to build, train and test the neural network models of a PUF and the MLP class is used to do the same with a MLP model.

Part B: Challenge-Response Pairs

The CRPs for this experiment are prepared and used as follows:

- Import the challeges and responses for each FPGA from Lab 1 into their respective directory.
- Make sure the challenges and responses have the correct format - a .npz file containing 50,000 8-bytes hex strings for challenges and a .npz file containing respective 50,000 16-bit binary strings for responses.
- The ChallengeResponseSet class from PartA is then used to load and split the CRP dataset into training/testing data for the models.

Part C: Gathering the results

Our procedure for Part C will take 4 steps:

Step 1: The pufModel class object and scikitlearn's MLP class object is used to train the neural network on the CRP data from Part B.
Step 2: The CRP data is split into training and testing sets based on ratio p.
Step 3: The split ratio p is gradually increased from 5% to 95% in increments of 5%.
Step 4: The accuracy of the models on the percentage of the testing data is observed and recorded for the different split ratios .

Part D: XOR PUF

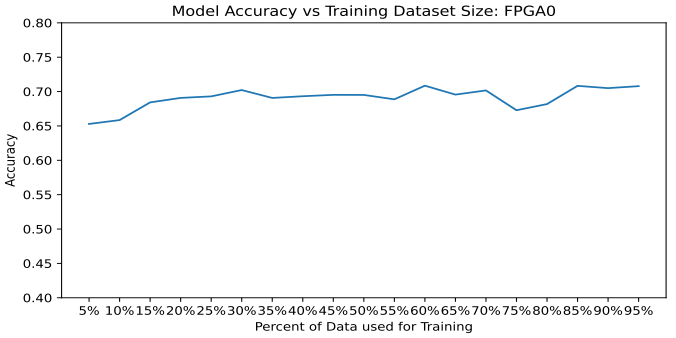
The experiment is repeated for 2-XOR PUF and 4-XOR PUF, respectively as follows:

Step 1: For 2-XOR PUF, the 16-bit responses from CRP dataset in Part B are XOR'd 2-bits at a time to have a total of 8-bits per response.
Step 2: Similarly, for 4-XOR PUF, the 16-bit responses from CRP dataset in Part B are XOR'd 4-bits at a time to have a total of 4-bits per response.
Step 3: Then Part C is repeated for the updated CRPs for the 2-XOR and the 4-XOR PUFs respectively.
Step 4: The accuracy of the model is observed and recorded for the different split ratios .

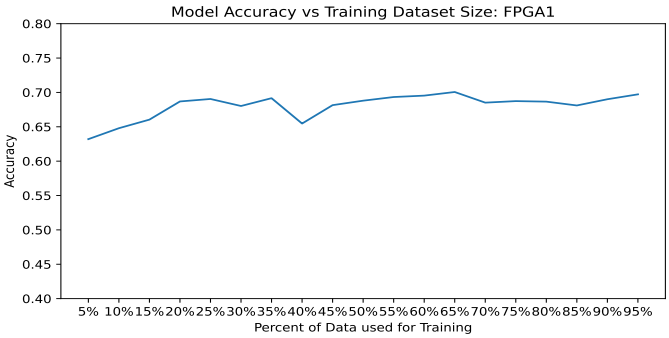
IV. Results

The results of this work are shown in Tables I and Figures 1-6 respectively. For each APUF, 1) Regular APUF, 2) 2BIT XOR APUF and 3) 4BIT XOR APUF, the accuracies of the Artificial Neural Network (ANN) Model and SciKitLearn's MLP model are shown.

Notably, the ANN's performance generally gets better with more training data on the APUF model, however there are drops in performance that prevent a linear relationship from forming. Observing the data, the model shows a trend across

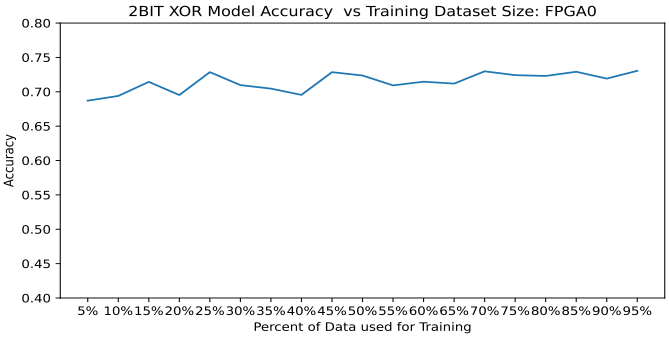


(a)

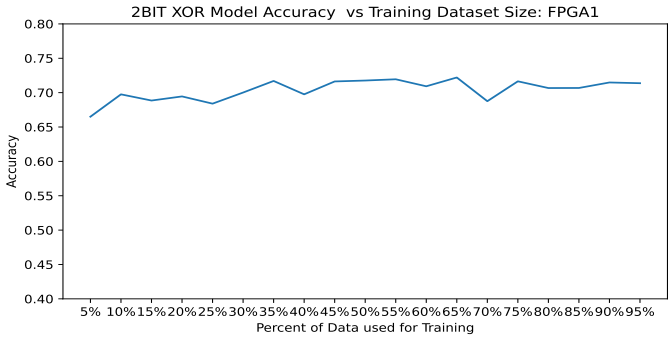


(b)

Fig. 1: Accuracies of the trained ML APUF for FPGA0 and FPGA1



(a)



(b)

Fig. 2: Accuracies of the trained ML 2-Bit XOR APUF for FPGA0 and FPGA1

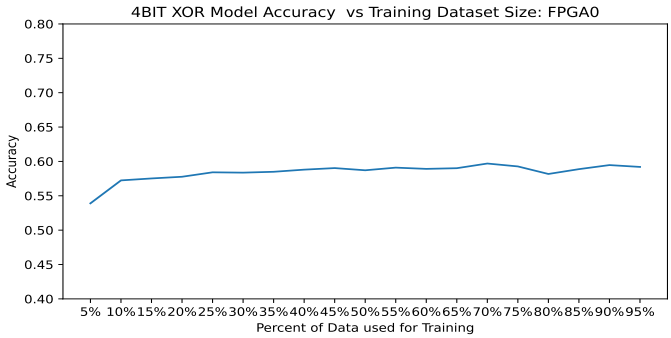
APUFs where when the percentage of training data is 65-70%, it achieves it's highest accuracy. When the APUF is transformed into a 2BIT XOR APUF, the model continues to perform similarly in learning the features. However, when the APUF is transformed into a 4BIT XOR APUF. The performance of the ANN sharply decreases.

With SciKitLearn's MLP model, the results are much more observably linear. Compared to the ANN, the MLP model shows a strong linear relationship that shows the MLP performance improves with more and more training data. Similarly, the MLP's performance sharply drops on the 4BIT XOR APUF model. Nevertheless, despite this linear relationship with the size of the training dataset, the maximum, average and minimum accuracies of the neural network are higher than the MLP's (Shown in Table 1).

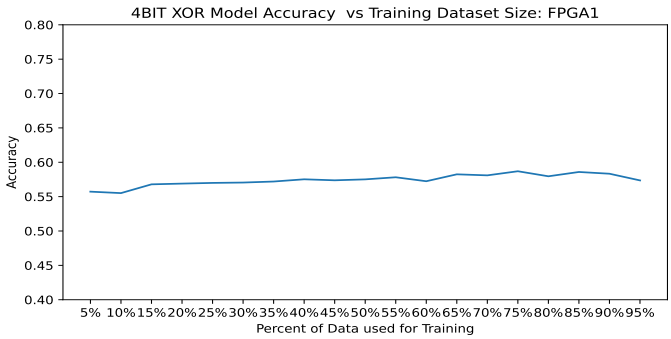
V. Discussion

These results show the relative power of neural networks compared to older machine learning techniques such as the MLP. The neural network achieves higher accuracy and with less required training data than the MLP model. We did not expect the accuracy of the neural network to be as variable as it was across the training/test splits, however we did expect there to be an improvement in accuracy as the model received more training data.

However, despite outperforming the MLP, the neural network in this experiment does not achieve extremely high accuracy in predicting the APUFs output. Ideally, one would

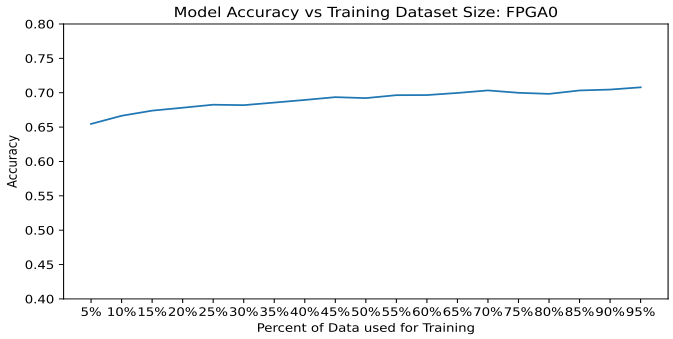


(a)

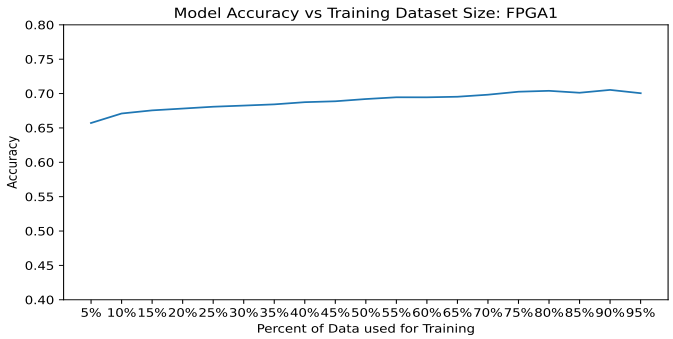


(b)

Fig. 3: Accuracies of the ML trained 4bit XOR APUF for FPGA0 and FPGA1

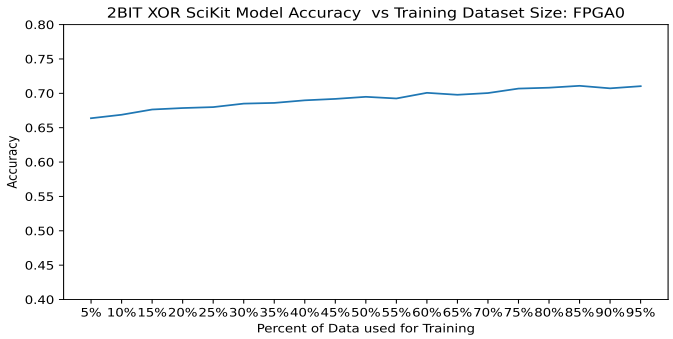


(a)

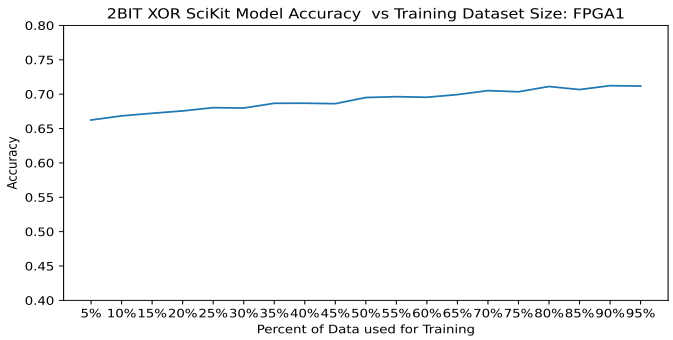


(b)

Fig. 4: Accuracies of the trained ML APUF for FPGA0 and FPGA1 using Scikit

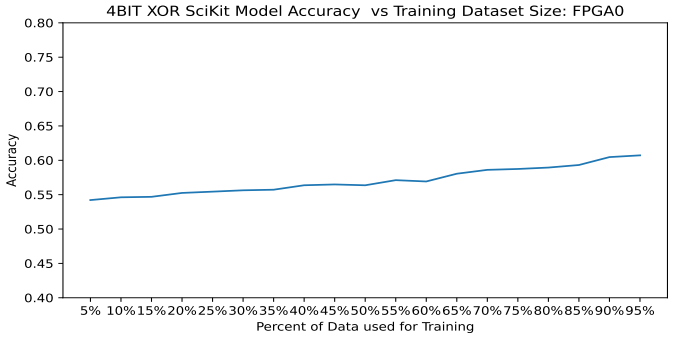


(a)

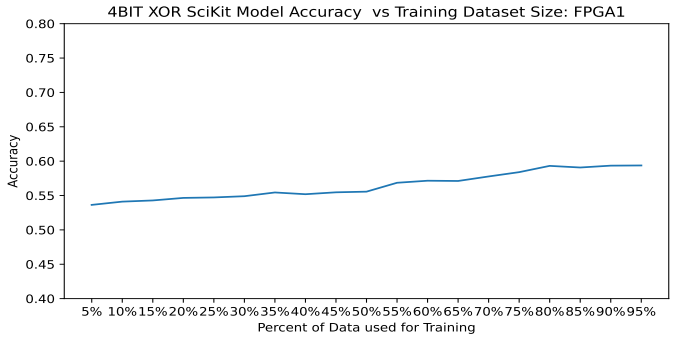


(b)

Fig. 5: Accuracies of ML based 2-Bit XOR APUF for FPGA0 and FPGA1 using Scikit



(a)



(b)

Fig. 6: Accuracies of ML based 4-bit XOR APUF for FPGA0 and FPGA1 using Scikit

TABLE I: Accuracy of Machine Learning Architecture on Various APUFs Across p Ratios

ML Arch.	APUF Model	Min.	Max.	Avg.	SD
ANN	APUF 1	0.645	0.717	0.691	0.0229
ANN	APUF 2	0.631	0.705	0.681	0.0199
ANN	2BIT XOR APUF 1	0.685	0.739	0.714	0.0018
ANN	2BIT XOR APUF 2	0.656	0.732	0.704	0.0012
ANN	4BIT XOR APUF 1	0.518	0.598	0.584	0.0039
ANN	4BIT XOR APUF 2	0.539	0.589	0.574	0.0018
MLP	APUF 1	0.655	0.716	0.690	0.0145
MLP	APUF 2	0.657	0.711	0.689	0.0131
MLP	2BIT XOR APUF 1	0.659	0.719	0.692	0.0002
MLP	2BIT XOR APUF 2	0.661	0.728	0.691	0.0009
MLP	4BIT XOR APUF 1	0.539	0.617	0.570	0.0018
MLP	4BIT XOR APUF 2	0.533	0.603	0.564	0.0015

expect the accuracy to be in the 90th percentile or higher but our neural network only achieved accuracies in the 70th percentile. This means that it would be good at breaking the APUF's security, but it does not completely destroy security properties.

During implementation of the experiment, instatiating and training the neural network went smoothly. However, there were some issues transforming the datasets into a suitable form for the MLP model. After further investigation and consultations with the TA, we learned that the dataset shape used for the neural network was also suitable for the MLP model and this allowed us to train the MLP model.

In future work, we would spend more time tweaking the neural network architecture. We would investigate how adding

temporal processing to the network would affect its learning and accuracy in model the various APUFs.

VI. Conclusion

In this lab, APUFs, 2-bit XOR APUFs, and 4-bit XOR APUFs were modeled using a Tensorflow based neural network and a ScikitLearn MLP model. The APUFs were modelled using a 50,000 CRP dataset obtained from previous work. The experiment was implemented via Python in Jupyter Notebooks running on an Ubuntu VM. The neural network modeled APUFs showed better performance with less accuracy, however its accuracy loosely trended with an increase in training data size. The neural network typically performed best with 65-70% of the dataset used for training. The MLP model on the other hand, showed a strong linear correlation with an increase in training size, but did not out perform the neural network. Ultimately, the authors learned how powerful neural networks are when compared to historic ML models and would like to explore how adding temporal features to the training process would further improve neural network performance.

References

[1] R. Karam, S. Katkoori, and M. Mozaffari-Kermani, "Experiment 2: Machine Learning Attack on PUF," in *Practical Hardware Security Course Manual*. University of South Florida, Aug 2022.