Department of Electrical and Computer Engineering
University of Thessaly

GLITCH ANALYSIS USING MACHINE LEARNING TECHNIQUES

Author: Christos Voutsadakis

Diploma Thesis

Supervisor: Georgios Stamoulis

Volos, August 2020

Department of Electrical and Computer Engineering
University of Thessaly

GLITCH ANALYSIS USING MACHINE LEARNING TECHNIQUES

Author: Christos Voutsadakis

Diploma Thesis

Supervisor: Georgios Stamoulis, Professor

Additional Committee Members:
Fotios Plessas, Associate Professor
Antonios Dadaliaris, Assistant Professor

Volos, August 2020

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Πανεπιστήμιο Θεσσαλίας

ΑΝΑΛΥΣΗ ΣΠΙΝΘΗΡΙΣΜΩΝ ΜΕ ΤΕΧΝΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

Χρήστος Βουτσαδάκης

Διπλωματική Εργασία

Επιβλέπων: Γεώργιος Σταμούλης, Καθηγητής

Μέλη Επιτροπής:
Φώτιος Πλέσσας, Αναπληρωτής Καθηγητής
Αντώνιος Δαδαλιάρης, Επίκουρος Καθηγητής

Βόλος, Αύγουστος 2020

Acknowledgements

I would like to thank professor Stamoulis for giving me the opportunity to work for my thesis on a topic I found highly interesting.

I would also like to thank doctoral candidate Dimitrios Garyfallou for his guidance and seamless cooperation.

A special thanks to Nikos Arvanitopoulos for his help in all things machine learning.

Lastly, I would like to thank my brother, my parents and my friends for their love and support throughout my academic years and beyond.

Abstract

CMOS circuits are a category of integrated circuits broadly used in a number of devices. A key factor in this is their low power consumption. A cause of elevated power consumption, or more accurately, dissipation, is a phenomenon known as a glitch. The power dissipated during glitches serves no functional purpose in the circuit, while the levels of this dissipation are high enough that a number of techniques for glitch elimination have been proposed. These techniques, however, may lead to unreasonably complicated circuits. For this reason, sometimes an attempt at predicting the level of power that is dissipated during a glitch is preferable, in order to account for it. The purpose of this thesis is to examine the use of machine learning techniques for the prediction of the output voltage and power supply current of a circuit during a glitch. To this end, a number of glitch SPICE simulations were used for the training of two machine learning models based on random forest regression. These models were then used to predict the values of the output voltage and the power supply current for similar glitch simulations. Experimental results on a two-input NAND gate implemented at 45 nm show that our prediction technique achieves an average mean error of 0.01 mV (0.001%) for the output voltage and 0.69 μA (15.64%) for the power supply current, compared to the respective values of the SPICE simulations. Therefore, there is a promising basis for further research on machine learning algorithms for the prediction of glitch behavior.

Περίληψη

Τα κυκλώματα CMOS είναι μια κατηγορία ολοκληρωμένων κυκλωμάτων ευρείας χρήσης σε πληθώρα συσκευών. Ένας σημαντικός παράγοντας σε αυτό είναι η χαμηλή κατανάλωση ισχύος τους. Μια αιτία αυξημένης κατανάλωσης, ή πιο συγκεκριμένα απώλειας, ισχύος είναι ένα φαινόμενο γνωστό ως σπινθηρισμός. Οι σπινθηρισμοί καταναλώνουν ισχύ χωρίς κάποιο όφελος για το κύκλωμα, ενώ η ποσότητα ισχύος που χάνεται κατά τους σπινθηρισμούς είναι τέτοια ώστε να έχει οδηγήσει σε αρκετές τεχνικές που αποσκοπούν στην εξάλειψή τους. Όμως αυτές οι τεχνικές πολλές φορές οδηγούν σε αδικαιολόγητα αυξημένη πολυπλοκότητα του κυκλώματος. Για τον λόγο αυτό, ενδέχεται κατά περίπτωση να είναι προτιμότερη η πρόβλεψη της ακριβής κατανάλωσης ισχύος, με σκοπό να ληφθεί υπόψη κατά τον σχεδιασμό του κυκλώματος. Αυτή η διπλωματική εργασία αποσκοπεί στην εξέταση της προοπτικής πρόβλεψης κάποιων τιμών τάσης και ρεύματος του κυκλώματος κατά τον σπινθηρισμό, με τη χρήση μηχανικής μάθησης. Προς αυτό το σκοπό, προσομοιώσεις σπινθηρισμών χρησιμοποιήθηκαν ως δεδομένα εκπαίδευσης δύο μοντέλων μηχανικής μάθησης, τα οποία βασίστηκαν σε έναν αλγόριθμο γνωστό ως οπισθοδρόμηση τυχαίου δάσους. Μετά την εκπαίδευση, τα μοντέλα χρησιμοποιήθηκαν για να προβλέψουν τις αντίστοιχες τιμές τάσης εξόδου και ρεύματος παροχής του δοκιμαστικού κυκλώματος κατά τη διάρκεια σπινθηρισμών. Τα αποτελέσματα που παρουσιάζονται στην παρούσα διπλωματική εργασία πηγάζουν από προσομοιώσεις με χρήση πύλης NAND 2 εισόδων σε τεχνολογία 45 nm και δείχνουν κατά μέσο όρο, μέσο σφάλμα 0.01mV (0.001%) για την τάση εξόδου και 0.69μA (15.64%) για το ρεύμα παροχής. Συνεπώς, δείχνουν υποσχόμενα για περαιτέρω έρευνα στο θέμα της χρήσης αλγορίθμων μηχανικής μάθησης με σκοπό την πρόβλεψη της συμπεριφοράς των σπινθηρισμών.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Motivation

This thesis is concerned with two different subjects and fields of study, namely glitch analysis and machine learning, and the use of the latter for progress in the former. Glitch analysis is a subject matter concerning unexpected circuit power consumption, while machine learning consists of self taught and self improving computer algorithms.

The goal of glitch analysis is the detection and reduction of occurrences of glitches, due to the high power dissipation they cause [1]. A number of different approaches have been proposed in the literature (four such approaches can be seen in [2, Ch. 2]), but none of them results in complete elimination of the phenomenon in conjunction with limited additional circuit complexity. Therefore, if glitches cannot be consistently avoided, an attempt should be made to predict their existence and the amount of power they are going to consume. This is where machine learning comes in.

1.2 Contribution

The aim of this thesis is to use machine learning, specifically an algorithm known as random forest regression, in order to gauge its effectiveness in accurate glitch predictions. The machine learning models use glitch simulations for training and their output was compared to the output of the SPICE simulation. The models were trained to predict the output voltage and the power supply current, respectively, during glitches in a simple two-input 45 nm NAND gate. The prediction of the output voltage had an average mean error of 0.01mV (0.001%), while the prediction of the power supply current had an average mean error of 0.69μA (15.64%), against the respective SPICE simulation values.

1.3 Outline

The second chapter is concerned with glitches and glitch power dissipation, while also providing some information on circuit simulation program SPICE.

The third chapter delves into machine learning, the algorithm used to create the models and the general theory of implementing a machine learning model.

The fourth chapter details the specific steps of the experiments that lead to this thesis' results.

The fifth chapter presents those results.

The sixth and final chapter comments on the results and proposes steps that build on them for future work.

Chapter 2

GLITCH ANALYSIS

2.1 Glitches and Power Consumption

CMOS stands for Complementary Metal-Oxide-Semiconductor. CMOS is the semiconductor technology used in the manufacturing of the transistors for most modern computer microchips. Therefore, digital logic circuits are also created with CMOS technology. Two remarkable characteristics of CMOS are its high noise immunity and low static power consumption [3].

There are two ways power is consumed or dissipated in CMOS: statically and dynamically. Static power consumption includes leakage currents and power required for the device to remain on standby, while dynamic power consumption is the power consumed while the circuit is active, including power dissipated during glitches. Dynamic power constitutes about 80% of the total power consumed by the circuit and glitch power dissipation can be between 20 and 70% of the total power [2, Ch. 1]. Consequently, there is a need to understand and limit the effects of glitches.

Glitches are unwanted transitions of a logic gate's output that have no functionality. They are momentary switches of the gate's output voltage, while it has no operational reason to change value. If the output of a logic gate is 1, a glitch would be the occurrence of an instantaneous spike to a value of 0 followed by a return to 1 [4]. This behavior could then, if the circuit designer had not taken relevant precautions, be propagated throughout the circuit, adding successively to the initial redundant power dissipation.

The problem is that glitch power dissipation can vary wildly and there is no reliable way to get an accurate prediction, since, in practice, each glitch can vary. This thesis proposes a way of predicting elements of glitch power dissipation, using as an example a simple NAND gate and applying a machine learning algorithm that attempts to predict the output voltage and the power supply current during the abnormality caused by the glitch.

2.2 Glitch Example

A two-input NAND logic gate (NAND2) is taken as an example to demonstrate how a glitch can be created. This gate has the following truth table:

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

figure 1: NAND2 truth table

If A and B have different values and both need to switch, say from 01 to 10, the output should remain 1 throughout. However, in practice there could be a brief moment where both A and B have a value of 1, therefore changing the output from 1 to 0 then back to 1. That would be a glitch, as seen by the plot of the output voltage in figure 2. The plots of course show that the values in actual circuits do not instantly change, but rather have a transition time, i.e. time to transition from low-to-high or high-to-low.

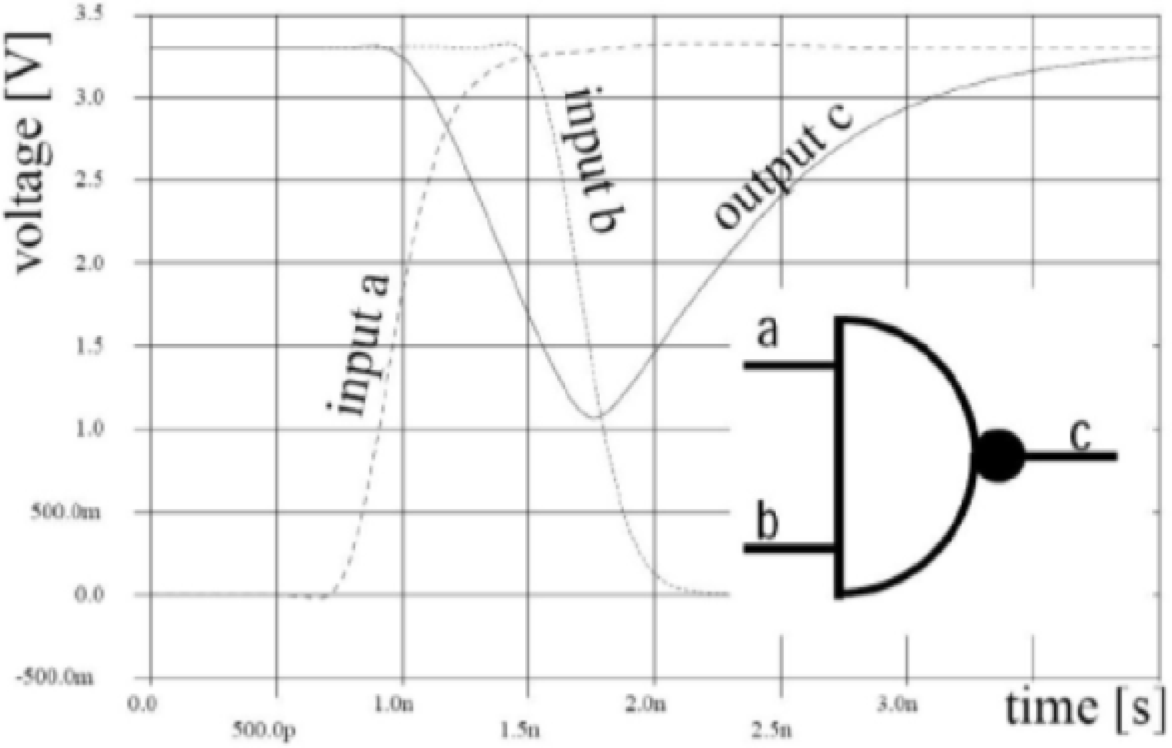

figure 2: NAND2 voltage values during a glitch

Chapter 3

MACHINE LEARNING AND RANDOM FOREST REGRESSION

Tom M. Mitchell, a renowned scientist on the field, defines machine learning as: “the study of computer algorithms that improve automatically through experience” [5].

What is meant by that, as elaborated further by Mitchell, is: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

3.1 Brief History

The idea of machines self improving through experience in their attempt to solve problems, can be seen as early as 1943. In that year, neurophysiologist Warren McCulloch and mathematician Walter Pitts published a paper where they modeled a brain and its neurons as a network of electrical circuits [6].

Seven years later the famous Turing Test was created but its namesake, pioneer mathematician Alan Turing [7]. Two years after that, in 1952, Arthur Samuel, who popularized the term “machine learning” [8], created a self educating computer program that played checkers [7].

The following decades saw a number of new algorithms being developed, as well as achievements like a machine playing tic-tac-toe [9], but the field was restrained by the technology of its time.

A watershed event occured in 1997, when Deep Blue, a chess-playing computer developed by IBM, beat the reigning world champion of chess, Garry Kasparov [7]. The decade after that saw renewed interest in the field, aided by the rapid improvement of computer capabilities, which culminated in a number of large businesses investing into major machine learning research teams and projects. Some of these include:

- AlexNet (2012): The winner of the ImageNet competition. ImageNet is a visual database and AlexNet is the name of the neural network that had the highest accuracy of correctly recognizing its pictures and videos. AlexNet cemented the value of GPUs in machine learning computations [10].

- Google Brain (2012): A Google research team that focuses on detecting visual patterns in images and videos.

- DeepMind (2014): A neural network that learns to play board games and simple video games, owned by Google.

- DeepFace (2014): A facial recognition neural network developed by Facebook.

- AlphaGo (2016): Created by the developers of DeepMind, AlphaGo is a program that in 2016 became the first machine to beat a professional Go player in a regular match and in 2017 beat the world number 1 Go player, Ke Jie. Go is seen as a much more complex board game than chess [11].

3.2 Random Forest Regression

The machine learning algorithm used for the creation of this thesis' models is known as random forest regression. The reason this algorithm was chosen will be explained in chapter 3.3. In order to better understand this algorithm, some basic concepts will now be explained, demonstrating its type, use and methodology.

- Supervised and unsupervised learning: A supervised algorithm learns by using a data set* that helps it understand the expected output values, whereas an unsupervised algorithm creates structure out of the input without any prior knowledge as to what the output should look like [12, p. 3]. Random forest regression is a supervised learning algorithm.

- Classification and regression: Supervised learning algorithms are categorized into classification and regression algorithms based on the desired output. If the aim is the assignment (classification) of data into a discrete number of categories, then it is a classification algorithm. If on the other hand, the output consists of continuous values, then it is a regression algorithm [12, p. 3]. As the name suggests, random forest regression falls into the regression category.

- Decision tree learning: A random forest algorithm utilizes decision trees. Decision trees reach answers to a problem by answering sequential questions. A simple case of a classification decision tree is shown below (figure 3), as discrete answers are better in demonstrating the basic concept. Using single decision trees is not advised, as they are computationally expensive to train and their results are heavily variable dependent.

* each entry of the data set or dataset (the two terms are used interchangeably) consists of two things: a number of input variables and their corresponding output variables, the values of which are dependent on the input.

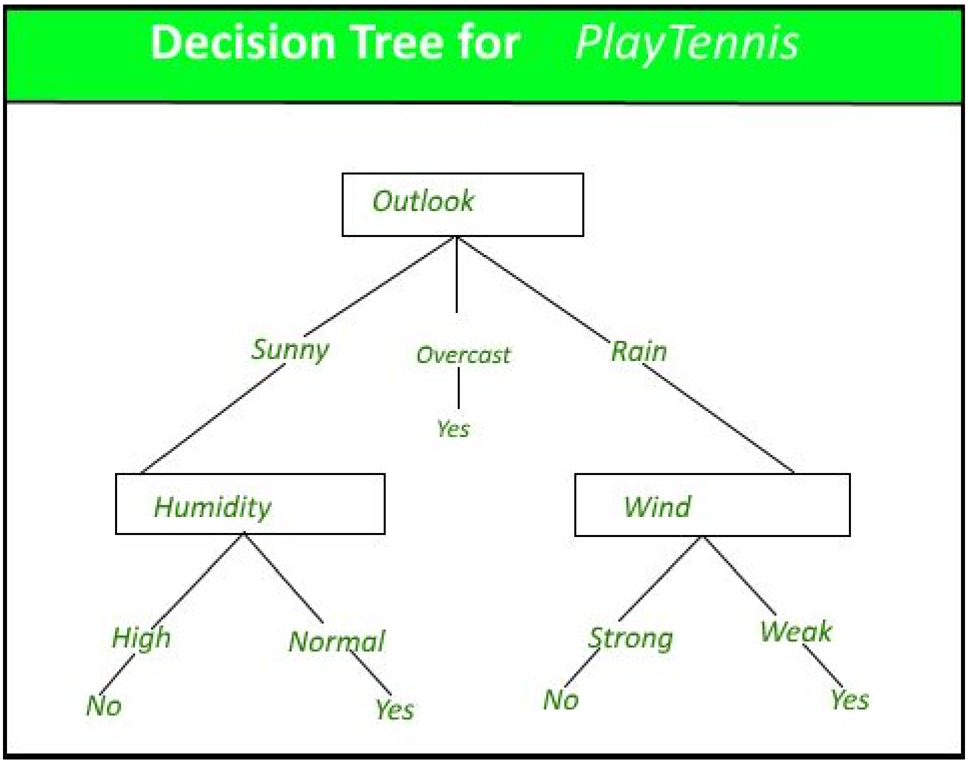

figure 3: Decision tree example. Source [13]

To sum up, random forest regression is a supervised regression algorithm that combines the predictions of multiple decision trees and accepts their mean as the more accurate prediction. This combination is why this algorithm is also categorized as an ensemble learning algorithm and also why it eliminates the weaknesses of single decision trees.

In a random forest algorithm, the trees run parallel to one another, with no interaction between them. The actual formula of the predictions is:

$\widehat{y} = \sum_{i=1}^{n} \left(\frac{1}{m} \sum_{j=1}^{m} W_j (x_i , x') \right) y_i$, with $\widehat{y}$ being the prediction for a new point x', n the size of the dataset, m the number of trees and $W_j (x_i , x')$ the weight of x' in relation to all of x_i which are all the other input variables. This is equal to $\frac{1}{k}$ for x_i on the same leaf but on the other trees, and 0 otherwise. [14]

3.3 Implementing Machine Learning Algorithms

The methodology followed for the creation of a model is based on Bishop's book [12]. This methodology can be followed for the implementation of machine learning algorithms in general. The steps after the dataset is created are the following:

1. Standardization: The first step is simplifying the dataset's input. This is done by normalizing it, that is, scaling the data to values between 0 and 1. This is done as follows:
   $y = \frac{x - min}{max - min}$, with x being the original value, y the new and min/max indicating the spectre of values. This is done for the entire input. The reason is to avoid variables with greater value span unduly affecting the importance of others. [12, p. 425]

2. Principal Component Analysis:
   The Principal Component Analysis (PCA) is used to detect variables that could be removed with minimal loss of information. PCA needs the data to be standardized. The first step is the computation of the covariance matrix of the input data, in order to determine the correlation between the variables. The covariance of two values X, Y is given by: $cov(X, Y) = E[(X - E[X])(Y - E[Y])]$, where E the mean value. The covariance matrix of 3 variables for example, is:

$$\begin{bmatrix} cov(X,X) & cov(X,Y) & cov(X,Z) \\ cov(Y,X) & cov(Y,Y) & cov(Y,Z) \\ cov(Z,X) & cov(Z,Y) & cov(Z,Z) \end{bmatrix}$$

   The important aspect of the covariances is not their value but their sign; a positive sign means that the two variables are correlated and a negative sign means they are inversely correlated.

   Next, the eigenvectors and eigenvalues of the covariance matrix are used to obtain the principal components of the original variables.

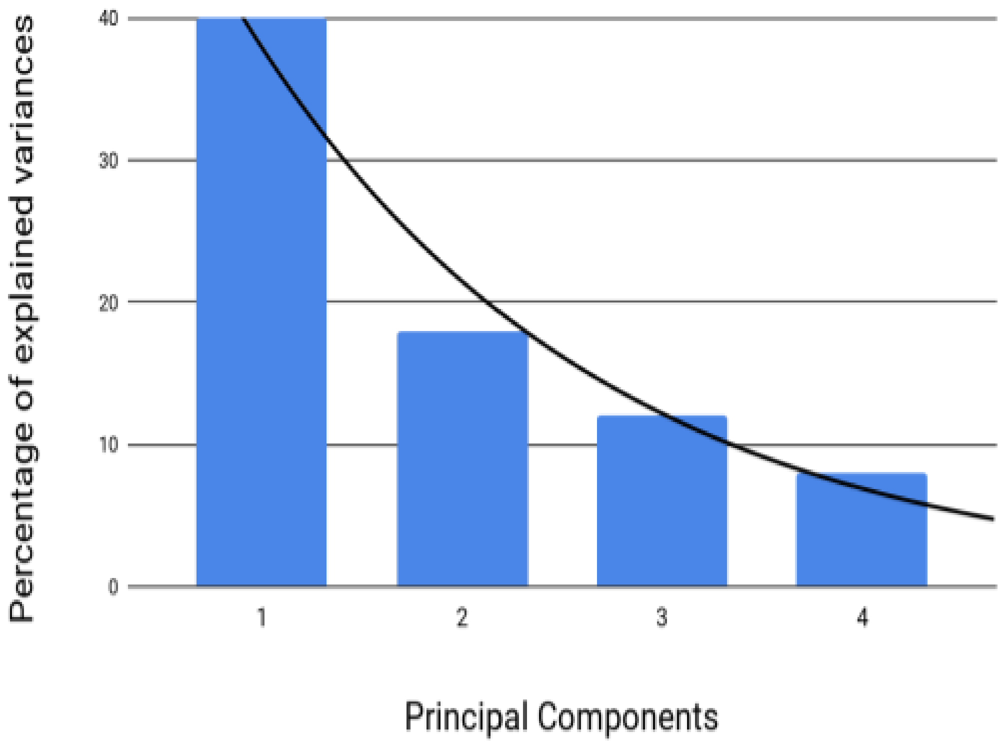

figure 4: Information stored in each successive principal component

Principal components are new variables created through combinations of the initial, in ways that create uncorrelated variables, with most of the information stored in the first one, as shown in figure 4, and less information in every subsequent variable.

The order of significance of the principal components is determined by sorting eigenvectors in order of their related eigenvalues, from highest to lowest. To ascertain the percentage of information each primary component carries, the relevant eigenvalue is divided by the sum of eigenvalues. A decision can then be made to remove primary components that carry a miniscule percentage, for the sake of lower complexity and speed. For a more detailed analysis of PCA, see [12, pp. 561-570].

3. Hyperparameter Optimization: model parameters like, for this specific algorithm, the number of decision trees in the forest or the depth of each tree, are known as hyperparameters.

   A method of hyperparameter optimization is the use of grid search. Grid search is an exhaustive search, wherein the model is trained on the dataset using different combinations of hyperparameters each time. Every combination is tested through cross validation; that is, the dataset is each time split into a number of subsets and the performance of each combination is the average of successful predictions across the subsets. The optimal hyperparameter combination is the one with the highest accuracy of predictions and that is the one chosen to be further analysed. [12, pp. 280-281]

4. Train-Test split: The entire dataset, both input and output, are then split into a "train" subset and a "test" subset. The "train" is the larger one, usually making up 80 or 90 per cent of the entire dataset. This is used, as the name suggests, for the model to train on, whereas the "test" part is used to test its accuracy. Usually repeated splits are needed to reach a more conclusive result. The train-test split method is preferred because it splits the dataset randomly, in an attempt to maintain its variety in values [12, p. 2].

5. Fit: By "fitting" the model, it is meant that the model trains on the "train" part of the dataset. With no further input from the programmer, the model trains by comparing predictions to real values and adjusting its inner weights assigned to every subsequent result. [12, p. 2]

6. Prediction: Predictions are then attempted when the model receives the new, "test" subset and uses its input values to predict the corresponding output.

7. <u>Mean Squared Error (MSE):</u> The mean squared error of the prediction of the “test” subset output measured against the actual “test” subset output, is used to gage the accuracy of the model. MSE is given by: $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2$, where n the number of values, Y_i the actual output and $\widehat{Y}_i$ the predicted output. Comparing the MSE of models that use different algorithms, a decision can be made as to the most effective algorithm [12, pp. 46-47].

Repeating steps 4 to 7 yields a more definitive result.

Chapter 4

PROPOSED APPROACH

4.1 Creating the Dataset

In order to create the dataset a number of simulations had to be run. These were done in HSPICE. HSPICE is one of the most accurate commercial continuations of the original SPICE [15], the established program of circuit simulations.

The circuit that was simulated was that of figure 5 and included a two-input, 45 nanometer NAND gate, two voltage sources for the input signals (V1, V2), a power supply source (Vdd) and a capacitor connected to the gate's output (C).

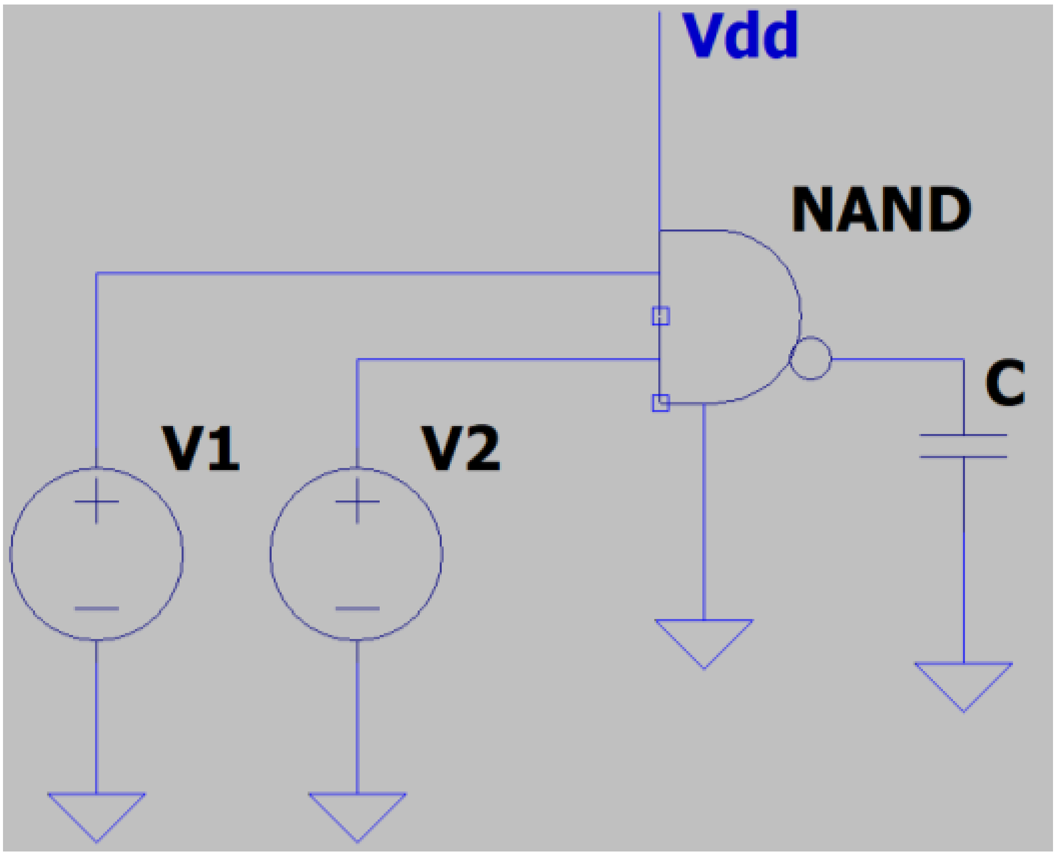

figure 5: Simplified schematic of the NAND2 circuit simulated

The method for creating the dataset of the size required in order to train the models, was built on the work done by A. O. Troumpoulou [16]. With the help of a C program, a number of transient analysis simulations with different variable values were run in an attempt to replicate numerous glitch cases.

The following four parameters were taken into account for the creation of glitches:

The first parameter was the capacitance value of the output capacitor.

Each simulation had two signals as input of the NAND2 gate. These two voltage sources were simulated as Piecewise Linear sources (PWL). This was done to set the actual points in time when the sources would start and end transitioning from 0 volts (V) to 1.10 V and from 1.10 V to 0 V respectively. The interval between the start and the end of this transition is known as transition time. The two transition times were two more of the aforementioned parameters. The actual points of time that the transitions begun and ended were variables set by the C program for each simulation. In every case however, one signal was at 0 V and 15 picoseconds (ps) later had a value of 1.10 V while the other started at 1.10 V and 15 ps later had a value of 0 V.

The fourth and final parameter was the distance between the transitions of the two signals. For this, the time interval between the points that each of the two signals was at 50% of the input voltage was measured.

To summarize, the four parameters were the capacitance value of the output capacitor (C), the two transition times (T1 and T2) and the distance between the two signals (HDIST).

There were 7 signal transition time values given by the C program and 22 capacitance values for the output capacitor. An example of input values during a conducted HSPICE transient analysis can be seen below.

```
V1 1 0 PWL 0PS 0V, 0.248169PS 0V, 0.496337PS 1.10V, 15PS 1.10V
V2 2 0 PWL 0PS 1.10V, 0.446704PS 1.10V, 0.694872PS 0V, 15PS 0V
C1 3 0 59.356700F


x_NAND2 7 5 2 3 1 NAND2_X1

.TRAN 1PS 200PS
```

figure 6: Example of HSPICE simulation input

As indicated by the underlined values in figure 6, in this example the first signal (V1) started transitioning at 0.248169 ps and finished at 0.496337 ps, while the second signal (V2) started at 0.446704 ps and completed its transition at 0.694872 ps. The difference between each set of underlined values is the corresponding transition time. It is also illustrated above that the output capacitor (C1) had a capacitance of 59.3567 fF. Finally, the initial transient analysis was performed for 200 ps with a timestep of 1 ps. This simulation time was selected in order to gage the time it would take for the output voltage to revert to its initial value of 1.10 V.

Every simulation produced a .LIS file with the results, whence the C program retrieved them one by one and saved them in two .TXT files, one for the current values, including the power supply current, and one for the output voltage values. An example of these .TXT files for one simulation can be seen below, with figures 7 and 8 having the values of its current variables and figures 11 and 12 of its voltage output variable.

TOT: 2760

C: 5.5646899999999997 T1: 0.078059600000000007 T2: 0.0171859 DIS: 0.011905687500000095 HDIS: 0.34457874999999988

	v6	c1	v0
0.	2.7139n	0.	5.8998n
1.00000p	4.2818u	-83.1182u	62.1859u
2.00000p	25.5011u	-87.8678u	96.8885u
3.00000p	-8.8866u	-7.4935u	-11.0620u
4.00000p	35.3169n	-11.4564u	4.9650u
5.00000p	22.0262u	-40.8684u	58.4029u
6.00000p	14.0925u	-22.6502u	33.7482u
7.00000p	-7.0893u	16.6548u	-25.4797u
8.00000p	-10.6083u	24.6009u	-36.3201u
9.00000p	6.5800u	-4.2171u	9.5671u
10.00000p	23.7683u	-33.0352u	55.4543u
11.00000p	18.5071u	-23.4959u	40.8988u
12.00000p	5.6429u	-965.8660n	5.8726u
13.00000p	-7.2214u	21.5642u	-29.1535u
14.00000p	-15.2370u	35.6895u	-51.0340u
15.00000p	27.7978u	-38.6762u	65.4938u
16.00000p	14.2882u	-15.2511u	28.8751u
17.00000p	4.2927u	2.0748u	1.7938u
18.00000p	4.2365u	2.1511u	1.6868u
19.00000p	4.1803u	2.2273u	1.5799u
20.00000p	4.1241u	2.3035u	1.4730u
21.00000p	4.0686u	2.3705u	1.3739u
22.00000p	4.0171u	2.3824u	1.3219u
23.00000p	3.9656u	2.3944u	1.2700u
24.00000p	3.9142u	2.4063u	1.2180u
25.00000p	3.8627u	2.4183u	1.1660u
26.00000p	3.8113u	2.4302u	1.1140u
27.00000p	3.7598u	2.4422u	1.0621u
28.00000p	3.7083u	2.4541u	1.0101u
29.00000p	3.6569u	2.4661u	958.1336n
30.00000p	3.6056u	2.4755u	908.3652n
31.00000p	3.5565u	2.4557u	884.3243n
32.00000p	3.5074u	2.4359u	860.2834n
33.00000p	3.4582u	2.4161u	836.2424n
34.00000p	3.4091u	2.3963u	812.2015n
35.00000p	3.3600u	2.3765u	788.1606n
36.00000p	3.3108u	2.3567u	764.1197n
37.00000p	3.2617u	2.3369u	740.0788n
38.00000p	3.2126u	2.3171u	716.0379n
39.00000p	3.1635u	2.2973u	691.9970n
40.00000p	3.1148u	2.2769u	668.7740n
41.00000p	3.0714u	2.2492u	655.1101n
42.00000p	3.0280u	2.2216u	641.4461n
43.00000p	2.9846u	2.1939u	627.7822n
44.00000p	2.9413u	2.1663u	614.1182n
45.00000p	2.8979u	2.1386u	600.4542n
46.00000p	2.8545u	2.1110u	586.7903n
47.00000p	2.8111u	2.0833u	573.1263n
48.00000p	2.7677u	2.0557u	559.4623n
49.00000p	2.7244u	2.0280u	545.7984n
50.00000p	2.6814u	2.0006u	532.4417n
51.00000p	2.6440u	1.9761u	522.6763n
52.00000p	2.6066u	1.9515u	512.9108n
53.00000p	2.5692u	1.9270u	503.1454n
54.00000p	2.5317u	1.9025u	493.3799n
55.00000p	2.4943u	1.8779u	483.6145n
56.00000p	2.4569u	1.8534u	473.8490n
57.00000p	2.4194u	1.8288u	464.0836n
58.00000p	2.3820u	1.8043u	454.3181n
59.00000p	2.3446u	1.7797u	444.5527n
60.00000p	2.3075u	1.7551u	435.0342n
61.00000p	2.2742u	1.7295u	428.4016n
62.00000p	2.2410u	1.7039u	421.7690n
63.00000p	2.2077u	1.6783u	415.1364n
64.00000p	2.1745u	1.6527u	408.5038n
65.00000p	2.1412u	1.6271u	401.8712n
66.00000p	2.1080u	1.6016u	395.2386n
67.00000p	2.0747u	1.5760u	388.6060n
68.00000p	2.0415u	1.5504u	381.9734n
69.00000p	2.0082u	1.5248u	375.3408n
70.00000p	1.9754u	1.4996u	368.7937n
71.00000p	1.9476u	1.4796u	363.2461n
72.00000p	1.9197u	1.4596u	357.6984n
73.00000p	1.8919u	1.4395u	352.1508n
74.00000p	1.8641u	1.4195u	346.6032n
75.00000p	1.8362u	1.3995u	341.0556n
76.00000p	1.8084u	1.3795u	335.5079n
77.00000p	1.7805u	1.3594u	329.9603n
78.00000p	1.7527u	1.3394u	324.4127n
79.00000p	1.7249u	1.3194u	318.8651n
80.00000p	1.6973u	1.2993u	313.4486n
81.00000p	1.6726u	1.2789u	309.5652n
82.00000p	1.6479u	1.2586u	305.6818n
83.00000p	1.6232u	1.2382u	301.7984n
84.00000p	1.5986u	1.2178u	297.9149n
85.00000p	1.5739u	1.1974u	294.0315n
86.00000p	1.5492u	1.1770u	290.1481n
87.00000p	1.5245u	1.1566u	286.2647n
88.00000p	1.4999u	1.1362u	282.3813n
89.00000p	1.4752u	1.1158u	278.4979n
90.00000p	1.4508u	1.0959u	274.6334n
91.00000p	1.4306u	1.0810u	270.9907n
92.00000p	1.4103u	1.0661u	267.3480n
93.00000p	1.3900u	1.0513u	263.7053n
94.00000p	1.3697u	1.0364u	260.0625n
95.00000p	1.3494u	1.0215u	256.4198n
96.00000p	1.3291u	1.0067u	252.7771n
97.00000p	1.3088u	991.7978n	249.1344n
98.00000p	1.2885u	976.9307n	245.4917n
99.00000p	1.2682u	962.0636n	241.8490n

figure 7: Simulation 2760
Current results, part 1

100.00000p	1.2481u	947.1397n	238.2980n
101.00000p	1.2300u	931.5518n	235.8189n
102.00000p	1.2119u	915.9639n	233.3398n
103.00000p	1.1938u	900.3759n	230.8607n
104.00000p	1.1757u	884.7880n	228.3816n
105.00000p	1.1575u	869.2001n	225.9025n
106.00000p	1.1394u	853.6122n	223.4234n
107.00000p	1.1213u	838.0243n	220.9443n
108.00000p	1.1032u	822.4363n	218.4652n
109.00000p	1.0851u	806.8484n	215.9861n
110.00000p	1.0673u	791.6573n	213.4986n
111.00000p	1.0527u	781.1036n	210.9136n
112.00000p	1.0380u	770.5499n	208.3286n
113.00000p	1.0234u	759.9962n	205.7436n
114.00000p	1.0088u	749.4424n	203.1585n
115.00000p	994.1958n	738.8887n	200.5735n
116.00000p	979.5817n	728.3350n	197.9885n
117.00000p	964.9676n	717.7813n	195.4035n
118.00000p	950.3536n	707.2276n	192.8184n
119.00000p	935.7395n	696.6738n	190.2334n
120.00000p	921.2337n	686.0235n	187.7225n
121.00000p	907.9930n	674.2440n	186.0776n
122.00000p	894.7524n	662.4646n	184.4328n
123.00000p	881.5117n	650.6851n	182.7879n
124.00000p	868.2711n	638.9056n	181.1430n
125.00000p	855.0305n	627.1262n	179.4982n
126.00000p	841.7898n	615.3467n	177.8533n
127.00000p	828.5492n	603.5672n	176.2084n
128.00000p	815.3085n	591.7878n	174.5635n
129.00000p	802.0679n	580.0083n	172.9187n
130.00000p	789.0479n	568.5864n	171.2513n
131.00000p	778.6063n	561.3436n	169.3211n
132.00000p	768.1648n	554.1008n	167.3909n
133.00000p	757.7232n	546.8580n	165.4607n
134.00000p	747.2817n	539.6152n	163.5305n
135.00000p	736.8401n	532.3723n	161.6003n
136.00000p	726.3986n	525.1295n	159.6701n
137.00000p	715.9570n	517.8867n	157.7398n
138.00000p	705.5155n	510.6439n	155.8096n
139.00000p	695.0739n	503.4011n	153.8794n
140.00000p	684.6918n	496.0264n	152.0141n
141.00000p	675.0046n	487.1103n	150.9074n
142.00000p	665.3174n	478.1941n	149.8006n
143.00000p	655.6301n	469.2780n	148.6938n
144.00000p	645.9429n	460.3619n	147.5870n
145.00000p	636.2557n	451.4458n	146.4803n
146.00000p	626.5685n	442.5296n	145.3735n
147.00000p	616.8813n	433.6135n	144.2667n
148.00000p	607.1940n	424.6974n	143.1600n
149.00000p	597.5068n	415.7813n	142.0532n
150.00000p	587.9988n	407.1903n	140.9155n
151.00000p	580.5858n	402.3995n	139.4162n
152.00000p	573.1727n	397.6086n	137.9170n
153.00000p	565.7596n	392.8178n	136.4178n
154.00000p	558.3466n	388.0269n	134.9185n
155.00000p	550.9335n	383.2361n	133.4193n
156.00000p	543.5204n	378.4453n	131.9200n
157.00000p	536.1074n	373.6544n	130.4208n
158.00000p	528.6943n	368.8636n	128.9215n
159.00000p	521.2812n	364.0727n	127.4223n
160.00000p	513.8913n	359.1222n	125.9827n
161.00000p	506.7713n	352.3056n	125.2405n
162.00000p	499.6513n	345.4890n	124.4983n
163.00000p	492.5313n	338.6724n	123.7561n
164.00000p	485.4113n	331.8558n	123.0139n
165.00000p	478.2913n	325.0392n	122.2717n
166.00000p	471.1713n	318.2225n	121.5294n
167.00000p	464.0513n	311.4059n	120.7872n
168.00000p	456.9313n	304.5893n	120.0450n
169.00000p	449.8113n	297.7727n	119.3028n
170.00000p	442.8402n	291.2563n	118.5241n
171.00000p	437.6093n	288.2481n	117.3194n
172.00000p	432.3785n	285.2400n	116.1146n
173.00000p	427.1476n	282.2318n	114.9099n
174.00000p	421.9168n	279.2237n	113.7051n
175.00000p	416.6859n	276.2155n	112.5004n
176.00000p	411.4551n	273.2073n	111.2956n
177.00000p	406.2242n	270.1992n	110.0908n
178.00000p	400.9934n	267.1910n	108.8861n
179.00000p	395.7625n	264.1829n	107.6813n
180.00000p	390.4622n	260.9343n	106.5291n
181.00000p	384.3504n	254.8767n	105.9913n
182.00000p	378.2385n	248.8191n	105.4535n
183.00000p	372.1267n	242.7614n	104.9157n
184.00000p	366.0149n	236.7038n	104.3779n
185.00000p	359.9030n	230.6462n	103.8400n
186.00000p	353.7912n	224.5885n	103.3022n
187.00000p	347.6793n	218.5309n	102.7644n
188.00000p	341.5675n	212.4733n	102.2266n
189.00000p	335.4557n	206.4156n	101.6887n
190.00000p	329.6184n	200.7731n	101.1198n
191.00000p	326.9902n	199.9811n	100.1874n
192.00000p	324.3619n	199.1891n	99.2549n
193.00000p	321.7336n	198.3972n	98.3224n
194.00000p	319.1054n	197.6052n	97.3900n
195.00000p	316.4771n	196.8132n	96.4575n
196.00000p	313.8489n	196.0213n	95.5251n
197.00000p	311.2206n	195.2293n	94.5926n
198.00000p	308.5923n	194.4374n	93.6601n
199.00000p	305.9641n	193.6454n	92.7277n
200.00000p	303.3358n	192.8534n	91.7952n

figure 8: Simulation 2760
Current results, part 2

At the top of figure 7, the input values for this specific simulation can be seen. In addition to the variables already mentioned, those of C (capacitance value of output capacitor), T1 and T2 (signal transition times) and HDIST (time interval between the moments when each of the two signals was at 50% of the input voltage), there is an additional DIS variable. This is the elapsed time between the moment that V2 started transitioning and the moment V1 finished transitioning. This variable was dropped during PCA, because its influence on the output was trivial.

Below those, there are four columns on the left that continue on the right and then continue similarly in figure 8. The leftmost column in each, with values from 0 to 200p, is the simulation time in ps. The other three columns are the corresponding current values for every picosecond, the output of the simulation: v6 indicates the power supply current, the one examined in this thesis, c1 the output capacitor current and v0 the ground current. In these three columns, ‘u’ indicates microampere, ‘n’ is nanoampere and ‘p’ is picoampere.

Upon further experimentation, it was decided to halve the step from 1 ps to 0.5 ps, so as to limit the chance of missing an upward or downward current spike, as seen in figure 9.

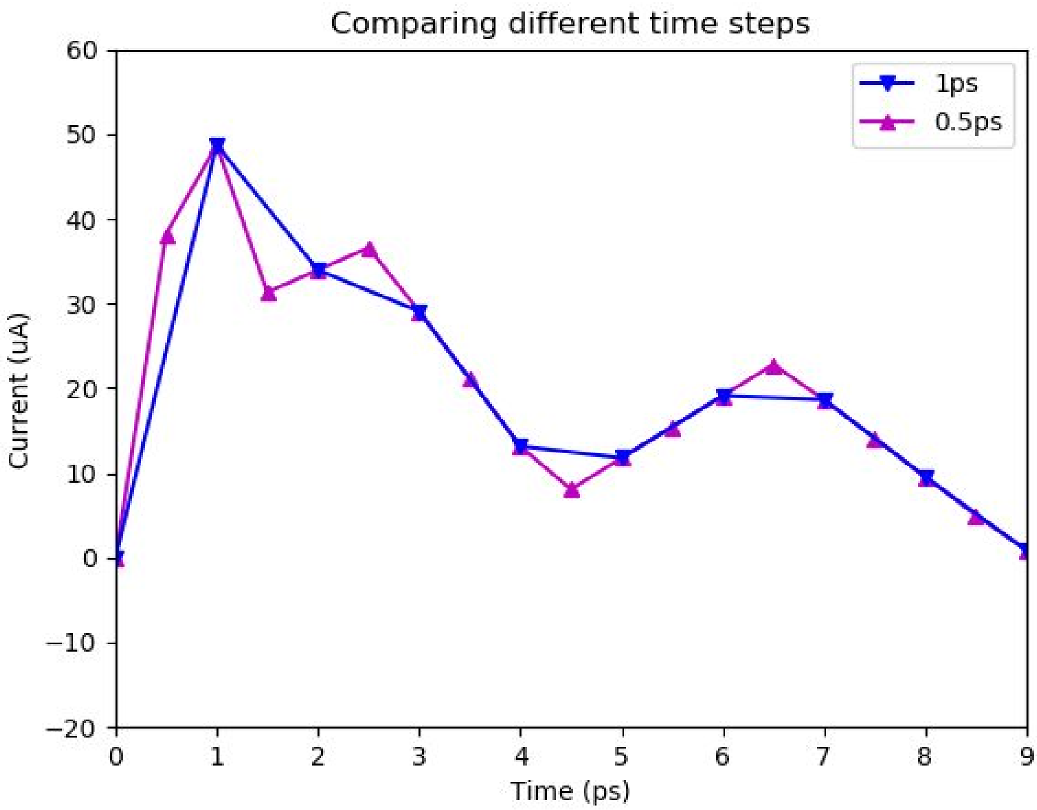

figure 9: Comparison of 0.5 ps and 1 ps time steps

Here the purple plot indicating a step of 0.5 ps, picked up on spikes that were missed with the blue plot of 1 ps step. It was also decided to set a cutoff point at 20 ps, as the current values were fairly stabilized from that point forward (see figure 10).

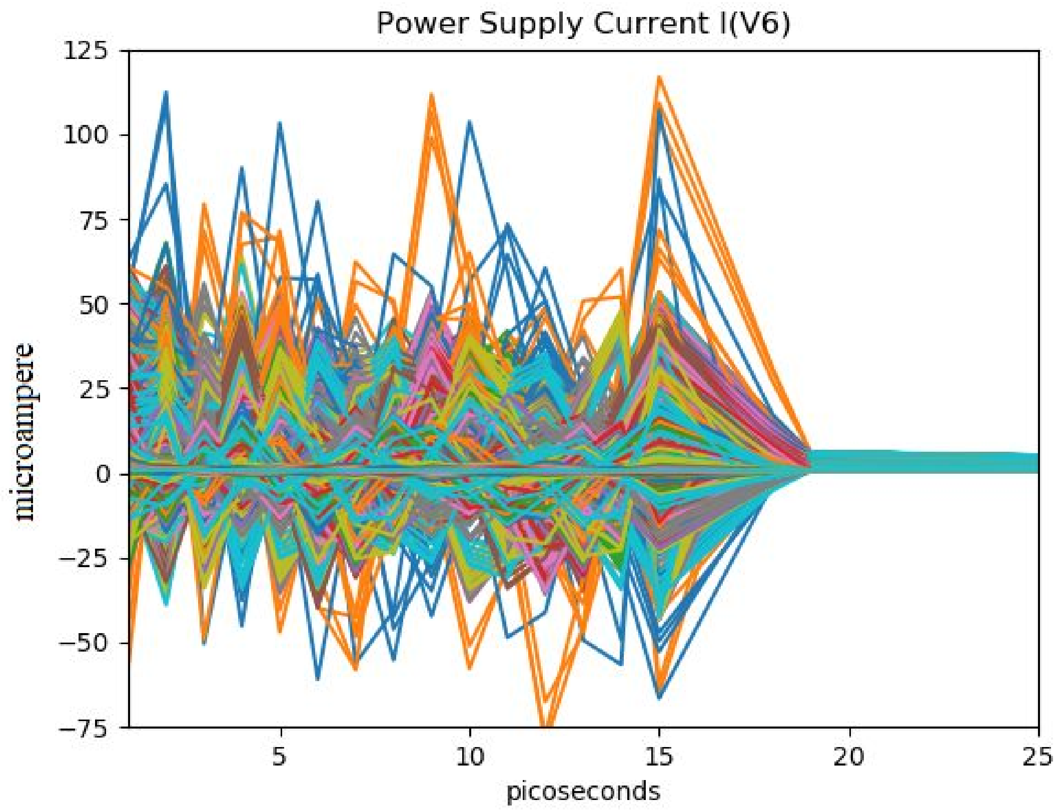

figure 10: Power supply currents of the entire dataset

Figures 11 and 12 continue the example of the .TXT values of simulation 2760, presenting its output voltage in two columns, the first being the time and the second the corresponding voltage values. Like before, the two columns start on the left of figure 11, continue on the right and then go on to figure 12 until the 200th picosecond.

```
TOT: 2760
C: 5.5646899999999997 T1: 0.07805960000000007 T2: 0.0171859 DIS: 0.011905687500000095 HDIST: 0.34457874999999988
            3
    0.          1.1000       50.00000p    1.0763
    1.00000p    1.0945       51.00000p    1.0766
    2.00000p    1.0815       52.00000p    1.0770
    3.00000p    1.0732       53.00000p    1.0773
    4.00000p    1.0686       54.00000p    1.0777
    5.00000p    1.0651       55.00000p    1.0780
    6.00000p    1.0633       56.00000p    1.0783
    7.00000p    1.0622       57.00000p    1.0787
    8.00000p    1.0615       58.00000p    1.0790
    9.00000p    1.0613       59.00000p    1.0793
   10.00000p    1.0611       60.00000p    1.0797
   11.00000p    1.0611       61.00000p    1.0800
   12.00000p    1.0613       62.00000p    1.0803
   13.00000p    1.0614       63.00000p    1.0806
   14.00000p    1.0616       64.00000p    1.0808
   15.00000p    1.0619       65.00000p    1.0811
   16.00000p    1.0623       66.00000p    1.0814
   17.00000p    1.0627       67.00000p    1.0817
   18.00000p    1.0631       68.00000p    1.0820
   19.00000p    1.0635       69.00000p    1.0823
   20.00000p    1.0639       70.00000p    1.0826
   21.00000p    1.0643       71.00000p    1.0829
   22.00000p    1.0647       72.00000p    1.0831
   23.00000p    1.0651       73.00000p    1.0834
   24.00000p    1.0656       74.00000p    1.0836
   25.00000p    1.0660       75.00000p    1.0839
   26.00000p    1.0664       76.00000p    1.0841
   27.00000p    1.0669       77.00000p    1.0844
   28.00000p    1.0673       78.00000p    1.0846
   29.00000p    1.0677       79.00000p    1.0849
   30.00000p    1.0682       80.00000p    1.0851
   31.00000p    1.0686       81.00000p    1.0853
   32.00000p    1.0690       82.00000p    1.0855
   33.00000p    1.0695       83.00000p    1.0858
   34.00000p    1.0699       84.00000p    1.0860
   35.00000p    1.0703       85.00000p    1.0862
   36.00000p    1.0707       86.00000p    1.0864
   37.00000p    1.0712       87.00000p    1.0866
   38.00000p    1.0716       88.00000p    1.0868
   39.00000p    1.0720       89.00000p    1.0871
   40.00000p    1.0725       90.00000p    1.0873
   41.00000p    1.0728       91.00000p    1.0875
   42.00000p    1.0732       92.00000p    1.0876
   43.00000p    1.0736       93.00000p    1.0878
   44.00000p    1.0740       94.00000p    1.0880
   45.00000p    1.0744       95.00000p    1.0882
   46.00000p    1.0748       96.00000p    1.0884
   47.00000p    1.0751       97.00000p    1.0886
   48.00000p    1.0755       98.00000p    1.0887
   49.00000p    1.0759       99.00000p    1.0889
```

figure 11: Simulation 2760
Voltage results, part 1

100.00000p	1.0891	150.00000p	1.0949
101.00000p	1.0893	151.00000p	1.0950
102.00000p	1.0894	152.00000p	1.0950
103.00000p	1.0896	153.00000p	1.0951
104.00000p	1.0897	154.00000p	1.0952
105.00000p	1.0899	155.00000p	1.0952
106.00000p	1.0900	156.00000p	1.0953
107.00000p	1.0902	157.00000p	1.0954
108.00000p	1.0904	158.00000p	1.0954
109.00000p	1.0905	159.00000p	1.0955
110.00000p	1.0907	160.00000p	1.0956
111.00000p	1.0908	161.00000p	1.0956
112.00000p	1.0909	162.00000p	1.0957
113.00000p	1.0911	163.00000p	1.0958
114.00000p	1.0912	164.00000p	1.0958
115.00000p	1.0913	165.00000p	1.0959
116.00000p	1.0915	166.00000p	1.0959
117.00000p	1.0916	167.00000p	1.0960
118.00000p	1.0917	168.00000p	1.0960
119.00000p	1.0919	169.00000p	1.0961
120.00000p	1.0920	170.00000p	1.0962
121.00000p	1.0921	171.00000p	1.0962
122.00000p	1.0922	172.00000p	1.0963
123.00000p	1.0923	173.00000p	1.0963
124.00000p	1.0924	174.00000p	1.0964
125.00000p	1.0926	175.00000p	1.0964
126.00000p	1.0927	176.00000p	1.0965
127.00000p	1.0928	177.00000p	1.0965
128.00000p	1.0929	178.00000p	1.0966
129.00000p	1.0930	179.00000p	1.0966
130.00000p	1.0931	180.00000p	1.0967
131.00000p	1.0932	181.00000p	1.0967
132.00000p	1.0933	182.00000p	1.0967
133.00000p	1.0934	183.00000p	1.0968
134.00000p	1.0935	184.00000p	1.0968
135.00000p	1.0936	185.00000p	1.0969
136.00000p	1.0937	186.00000p	1.0969
137.00000p	1.0938	187.00000p	1.0969
138.00000p	1.0939	188.00000p	1.0970
139.00000p	1.0940	189.00000p	1.0970
140.00000p	1.0941	190.00000p	1.0971
141.00000p	1.0942	191.00000p	1.0971
142.00000p	1.0942	192.00000p	1.0971
143.00000p	1.0943	193.00000p	1.0972
144.00000p	1.0944	194.00000p	1.0972
145.00000p	1.0945	195.00000p	1.0973
146.00000p	1.0946	196.00000p	1.0973
147.00000p	1.0946	197.00000p	1.0973
148.00000p	1.0947	198.00000p	1.0974
149.00000p	1.0948	199.00000p	1.0974
150.00000p	1.0949	200.00000p	1.0974

figure 12: Simulation 2760
Voltage results, part 2

In summary, the data utilized for this experiment consisted of **10780 simulations** of different combinations of the 4 variables (or machine learning features) mentioned above. Lastly, as can be seen in figures 7, 8, 11 and 12, each simulation produced 200 instances of values. Only 40 of those (0.5 step to 20ps) were used for the current, for the reason mentioned previously.

4.2 Creating the Models

In order to apply the relevant machine learning techniques elaborated upon in chapter 3, a number of tools had to be used. Firstly, version 2019.3.3 x64 of PyCharm, which is a Python language integrated development environment. The version of Python was Python 3.

Regarding libraries, pandas [17], Scikit-learn [18], Matplotlib [19] and Pickle [20] were employed. pandas enabled the reading and formatting of the data, whereas Scikit-learn provided all the machine learning tools and algorithms. Matplotlib was used as an efficient way to get a visual representation of the results. Pickle allowed the encoding of the models and their data standardization scalers into binary files and can also be used to load them from the binary files for use.

In order to be accessible to the methods presented by pandas, the data had to be transformed into .CSV format. A small Python program was created to that effect. Both initial .TXT files were split into two .CSV files each, one file containing the machine learning input and the other containing the output. Only the values for the current of the power supply were kept in the relevant .CSV file, as this was the current that was examined in this thesis. Figures 13 and 14 show a part of both .CSV voltage files, input and output.

```
C,T1,T2,DIS,HDIST
0.365616,0.0011737799999999999,0.0011737799999999999,0.0029344500000000051,0.24743513749999993
0.365616,0.0011737799999999999,0.0011737799999999999,0.0026410050000000018,0.24772858249999993
0.365616,0.0011737799999999999,0.0011737799999999999,0.0023475599999999985,0.24802202749999994
0.365616,0.0011737799999999999,0.0011737799999999999,0.0020541149999999953,0.24831547249999994
0.365616,0.0011737799999999999,0.0011737799999999999,0.001760669999999992,0.24860891749999994
0.365616,0.0011737799999999999,0.0011737799999999999,0.0014672249999999887,0.24890236249999992
0.365616,0.0011737799999999999,0.0011737799999999999,0.0011737799999999854,0.24919580749999992
0.365616,0.0011737799999999999,0.0011737799999999999,0.0008803349999999821,0.24948925249999995
0.365616,0.0011737799999999999,0.0011737799999999999,0.0005868899999997882,0.24978269749999996
0.365616,0.0011737799999999999,0.0011737799999999999,0.0002934449999997553,0.25007614249999999
```

figure 13: Some voltage input values in CSV format

```
V3-1,V3-2,V3-3,V3-4,V3-5,V3-6,V3-7,V3-8,V3-9,V3-10,V3-11,V3-12,V3-13,V3-14,V3-15,
1.0712,1.0208,0.9904402,0.9745031,0.9681033,0.9674427,0.9691912,0.9735976,0.97803
1.0712,1.0204,0.9908982,0.9740926,0.9682758,0.9673012,0.9692811,0.9735951,0.97806
1.0712,1.0202,0.9911403,0.9737543,0.9683532,0.9671658,0.9693122,0.9735645,0.97806
1.0712,1.0200,0.9912777,0.9735541,0.9684114,0.9670964,0.9693444,0.973559,0.978080
1.0712,1.0200,0.9913126,0.9734724,0.9684293,0.9670672,0.9693516,0.9735528,0.97808
1.0712,1.0200,0.9912592,0.9735176,0.9684156,0.9670854,0.9693408,0.973552,0.978076
1.0711,1.0181,0.9892419,0.9757521,0.9675263,0.9677629,0.9696531,0.9734098,0.97852
1.0711,1.0181,0.989321,0.9757162,0.9675734,0.9677537,0.9696675,0.9733991,0.978523
1.0711,1.0178,0.9897555,0.9754938,0.9678944,0.9676415,0.9697418,0.9733265,0.97850
1.0712,1.0171,0.9905877,0.9749687,0.9684588,0.9674193,0.969889,0.9732184,0.978485
```

figure 14: Some voltage output values in CSV format

To conform to the .CSV format, each simulation's variables in the input file and results in the output file, were written in one line and separated by commas. For both the voltage and the current, the initial value was dropped from the .CSV, given that it was a constant value of 1.10 V and 2.7139 nA respectively; as constants they were immaterial for predictions.

Scikit-learn provided both the machine learning algorithms and the tools required in order to train and test the models that were created:

- Standardization**:** The sklearn.preprocessing.StandardScaler class was used to standardize the data.

- Principal Component Analysis**:** sklearn.decomposition.PCA was a class imported to test the standardized weights calculated using the previous class. By testing the weights, it was concluded that the DIS variable had minimal effect on the outcome, therefore it was eliminated from the dataset.

- Train-Test split**:** sklearn.model_selection.train_test_split was used in order to split the dataset into two unequal portions in a 90-10 division. The larger was used to train the model, whereas the smaller was used to test the effects of the training.

- Random Forest Regression**:** sklearn.ensemble.RandomForestRegressor was the machine learning algorithm that was chosen, after a number of algorithms, including k-nearest neighbours, were tried**.** Details about random forest regression were outlined in section 3.2.

- Grid Search**:** sklearn.model_selection.GridSearchCV was used to determine some of the hyperparameters which are the parameters of the RandomForestRegressor class. Specifically it was determined that the best number of trees in the forest was 150.

- Mean Squared Error (MSE): sklearn.metrics.mean_squared_error provided the MSE of the model predicting the output of the train and of the test subsets. The requirements are a small difference between the two values and a small MSE on the test subset. The smaller this value is, the more accurate the predictions during the testing were. This step decided the use of random forest regression, since its MSE was consistently lower than that of k-nearest neighbours, the algorithm with the second lowest MSE.

- Mean Absolute Error (MAE): sklearn.metrics.mean_absolute_error gave an easier to understand metric, with mean absolute error being the prediction's deviation from the real values given in the actual units; V and μA respectively.

- Multi-output Regression: Finally, since the subject of this thesis is a problem of multi-output nature and regular machine learning algorithms tend to be single output oriented, a wrapper class had to be used in conjunction with sklearn.ensemble.RandomForestRegressor. This wrapper was sklearn.multioutput.MultiOutputRegressor.

After the algorithm was decided on and the models created, the dataset was repeatedly split into "train" and "test" and the input each time was standardized. Then, the models trained on the appropriate subdataset and attempted to predict the output of the "test" input, comparing the MSE of the two. Finally, the actual output and the predicted output were plotted on the same graph, utilizing the pyplot module of Matplotlib.

4.3 Model Application

The Pickle library was used in order to save the two models and also the two scalers responsible for standardizing the input into binary form. Pickle can be used to access them after the creation and use them to predict output voltage and power current supply during a glitch, given 4 values that represent: the capacitance of the output capacitor in femtofarads, the two transition times of the signals in picoseconds and the time interval between the moments each of the two was at 50% of the input voltage value, also in picoseconds.

Therefore, after the creation of the models and the respective scalers, the product of this thesis can be used with the help of a single python library: Pickle. The program that interfaces with the user simply loads the four binary files with the help of this library, uses the scalers and the models on the input provided by the suer and produces two vectors. The one is the output voltage values every picosecond and the other the values of the power supply current every half picosecond.

Chapter 5

RESULTS

5.1 Output Voltage

The prediction of the behavior exhibited by the gate's output voltage of the specific dataset was fairly accurate. Firstly, the MSE of the "test" subset predictions, compared to the actual "test" output has an average value of 0.00000003, out of 10 train-test splits. It is easily deductible therefore that the predictions made tend to be of high accuracy. The average MAE, the average actual deviation from the real values, was **0.00011 V** (average mean relative error of 0.001%).

Figures 15 to 18 illustrate comparisons between predicted and real output voltage values.

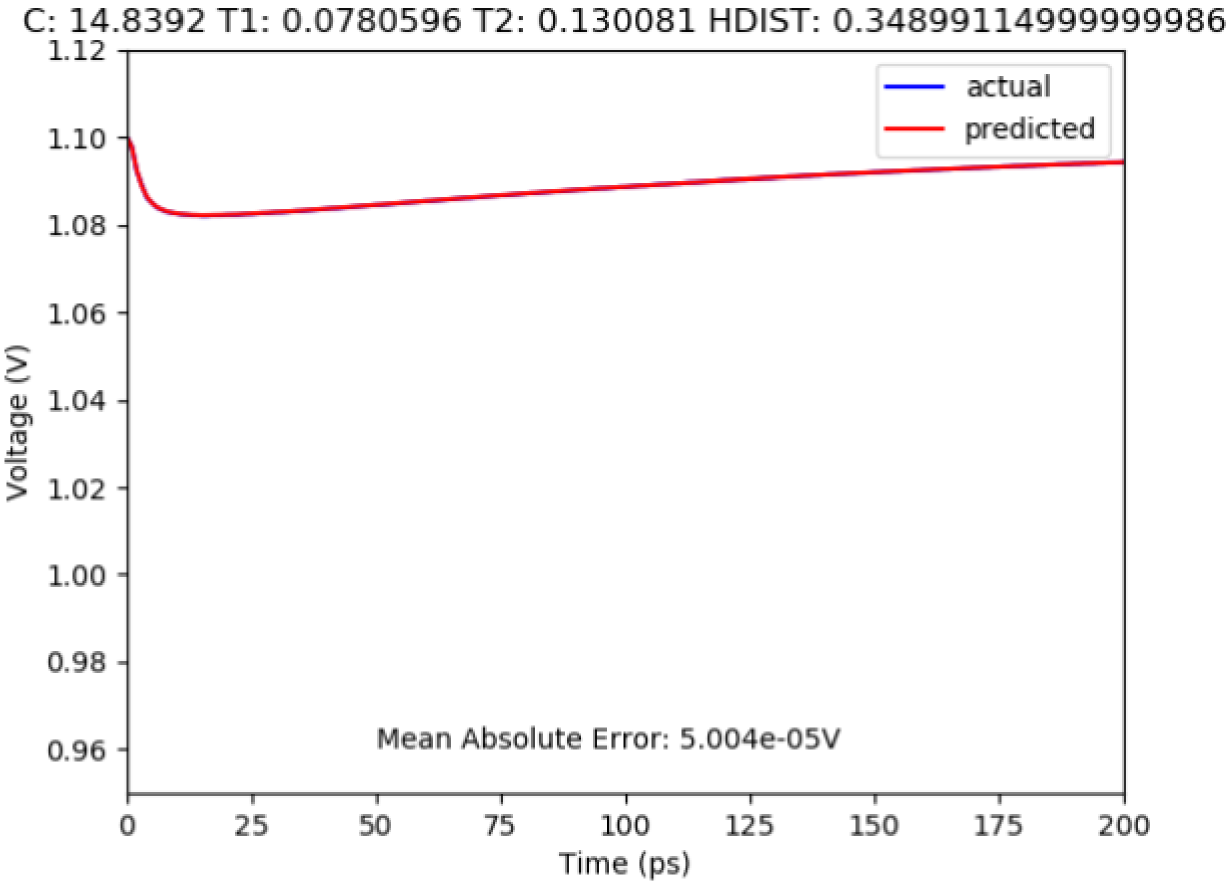

figure 15: A simulation with minimal voltage fluctuation

This example showcases a simulation where, while a glitch did occur, the output voltage did not decrease significantly from its original value. The red prediction plot and the blue plot of simulation values are identical in this scale.

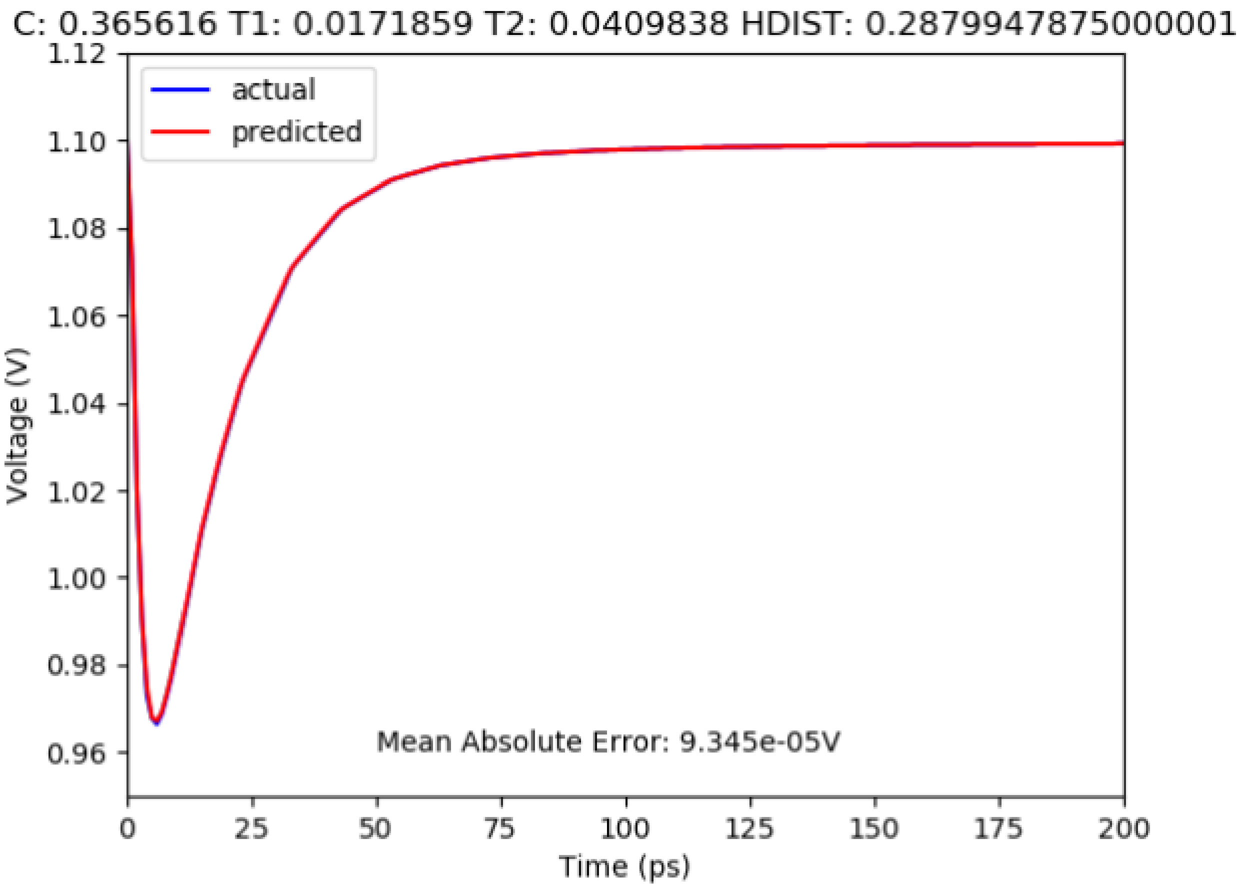

figure 16: A simulation with substantial voltage fluctuation

Same situation of identical values, but in this case the output voltage dropped substantially. A zoomed in view of the above graph follows, that demonstrates the existence of a slight deviation.

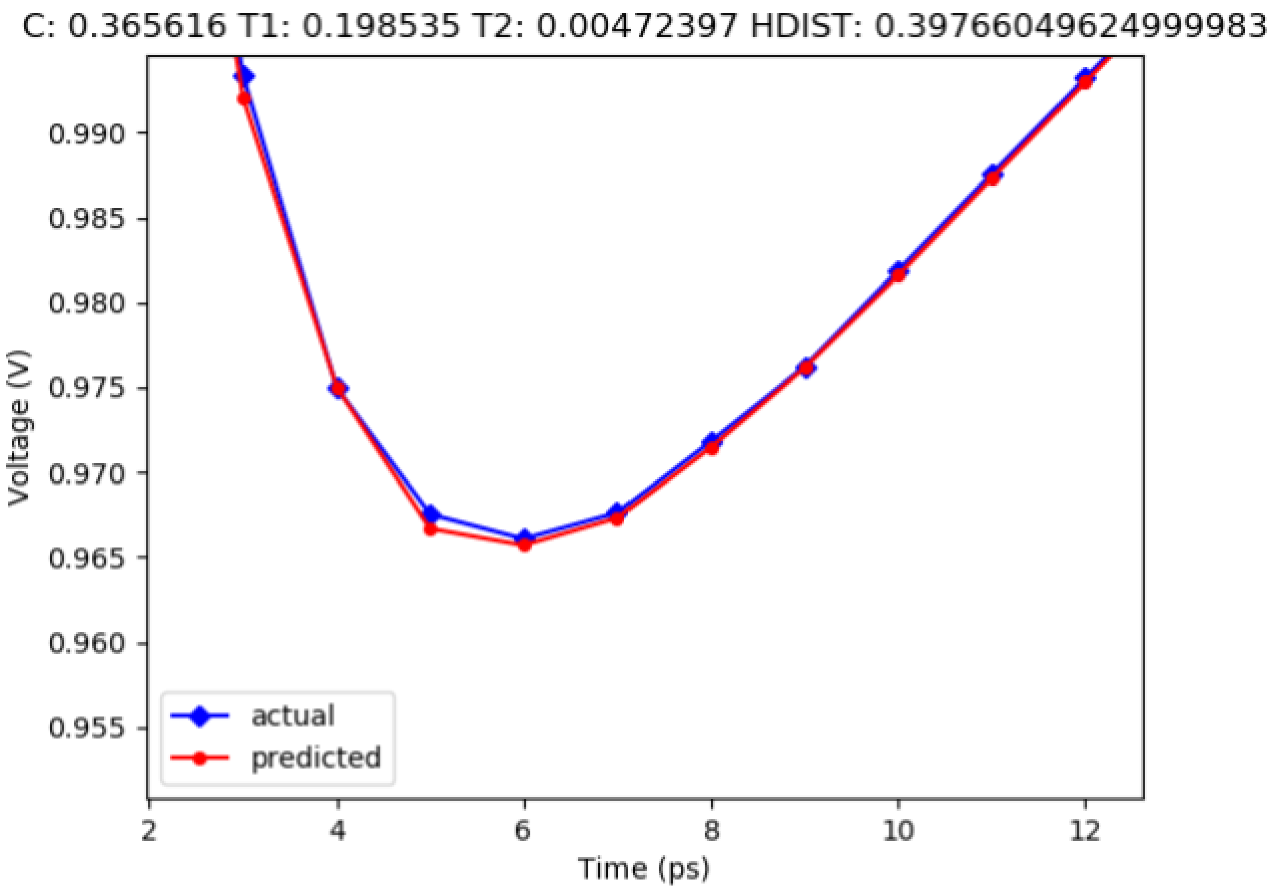

figure 17: A zoomed in view of the above example

Indeed here the slight deviation between actual and predicted values becomes apparent. Since the values in question are examined in V (as opposed to mV for example), this deviation is not a significant problem.

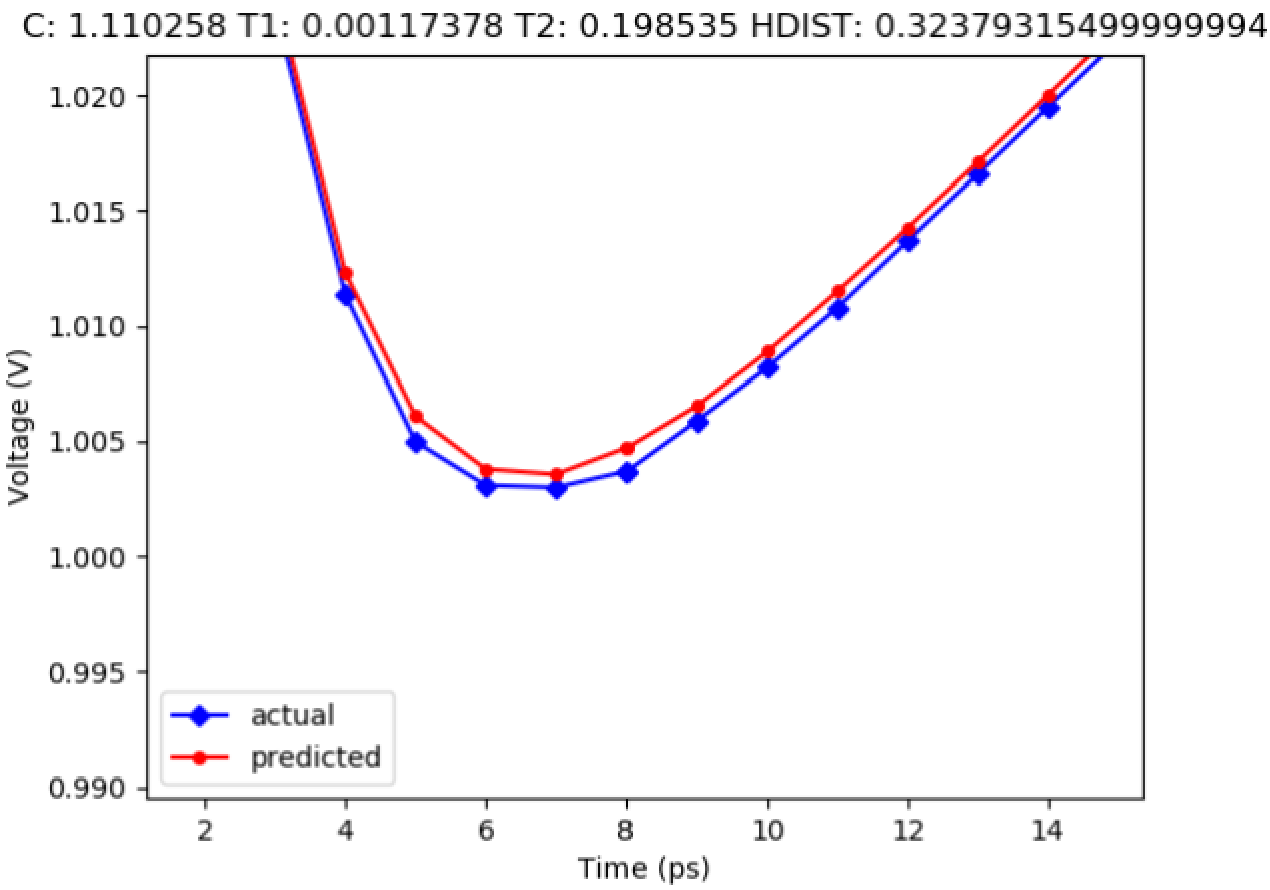

figure 18: A zoomed in view of another example

As this is a machine learning algorithm, complete and consistent accuracy is not the expected outcome; the goal is an approximation of the real values. Even with the possible deviations that may occur between predicted and real values as demonstrated in figure 18, an average MAE of 0.00011 V was deemed sufficient for the purposes of this thesis.

5.2 Power Supply Current

The results of the model predicting the power supply current present a wider range in accuracy. But due to the erratic behavior exhibited by the power supply current during a glitch, as was demonstrated in figure 10, such results are not unexpected. Even so, the model that was implemented achieved fairly accurate predictions.

Ten train-test splits of the dataset led to an average MSE of 5.17 and a best case MSE of 3.33. The specific test subset had 1078 simulations in it, 10% of the entire dataset. The average MAE of the predictions against these 1078 simulations was **0.69 μA** (average mean relative error of 15.64%). Using the model with the best MSE, 92.39% of the predictions had a MAE of less than 2 μA, 5.47% had a MAE of between 2 and 5 μA and only 2.13% had a MAE above 5 μA.

Examples follow (figures 19-23):

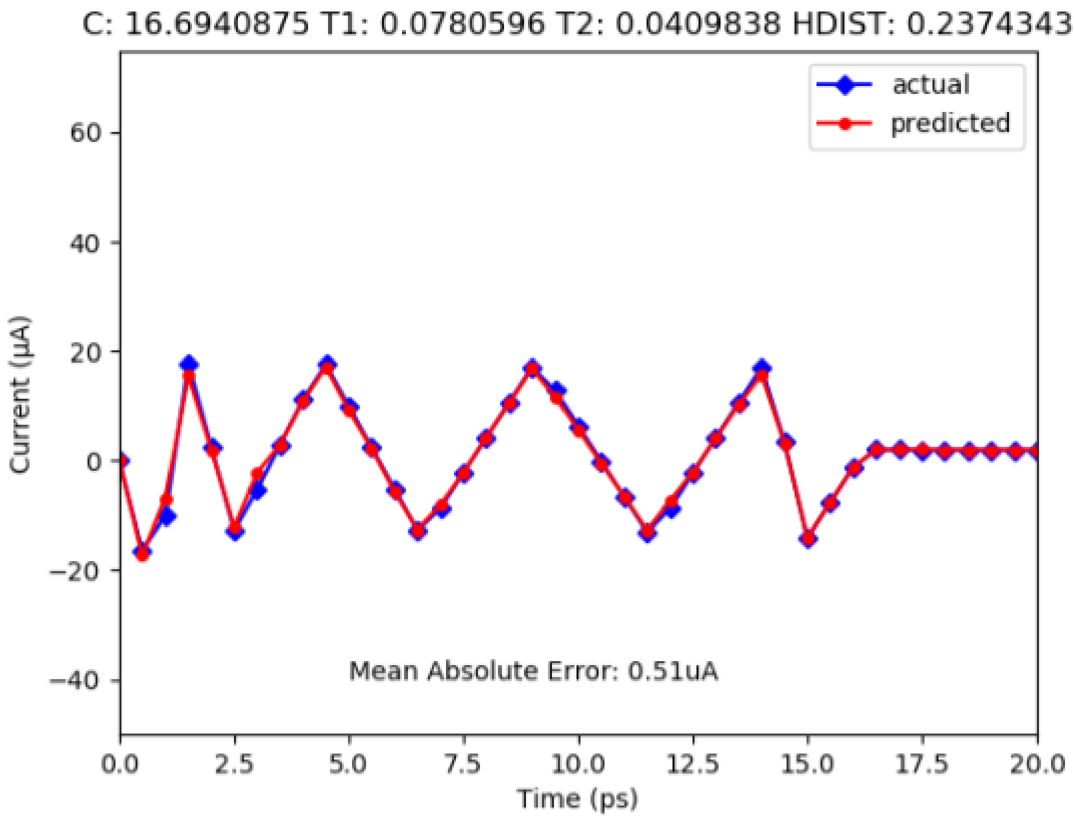

figure 19: An example of great accuracy (exhibited in 92.39% of cases)

The accuracy presented here is high. There were cases that the MAE was as small as 0.07 μA, as can be seen in the following page.

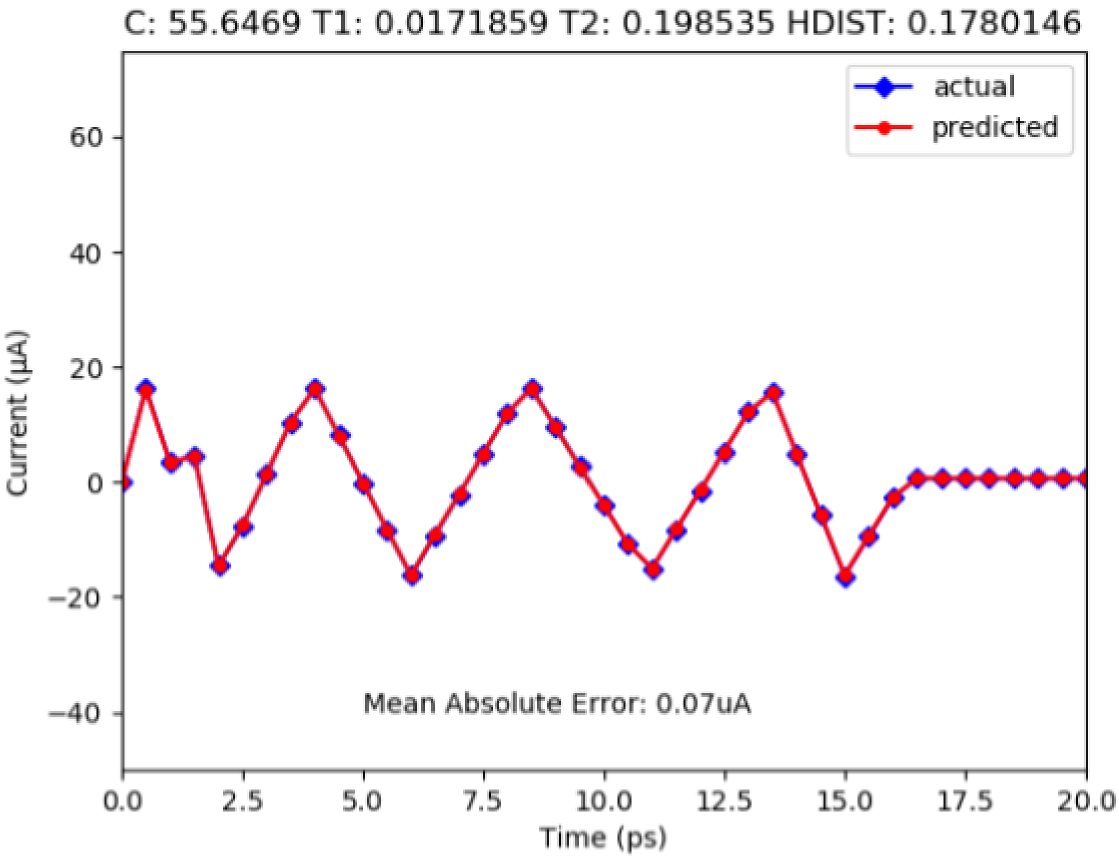

figure 20: Another similar, even more accurate, example

The difference between actual and predicted values here is almost non existent. The red plot seems like a carbon copy of the blue.

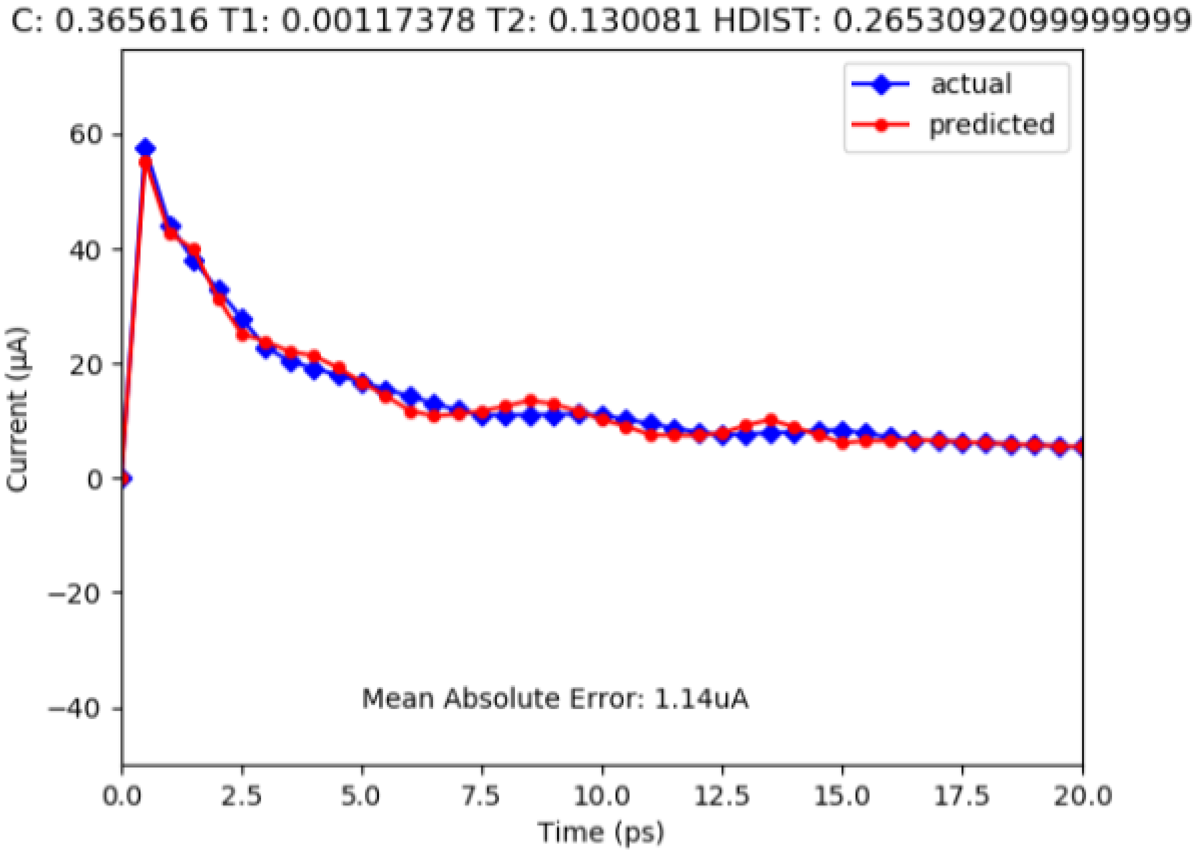

figure 21: A different case of similar accuracy

Not all cases have a similar kind of plot however, as was shown in figure 10. The model created, as evident here, is able to predict the power supply current to reasonable accuracy in a number of different cases.

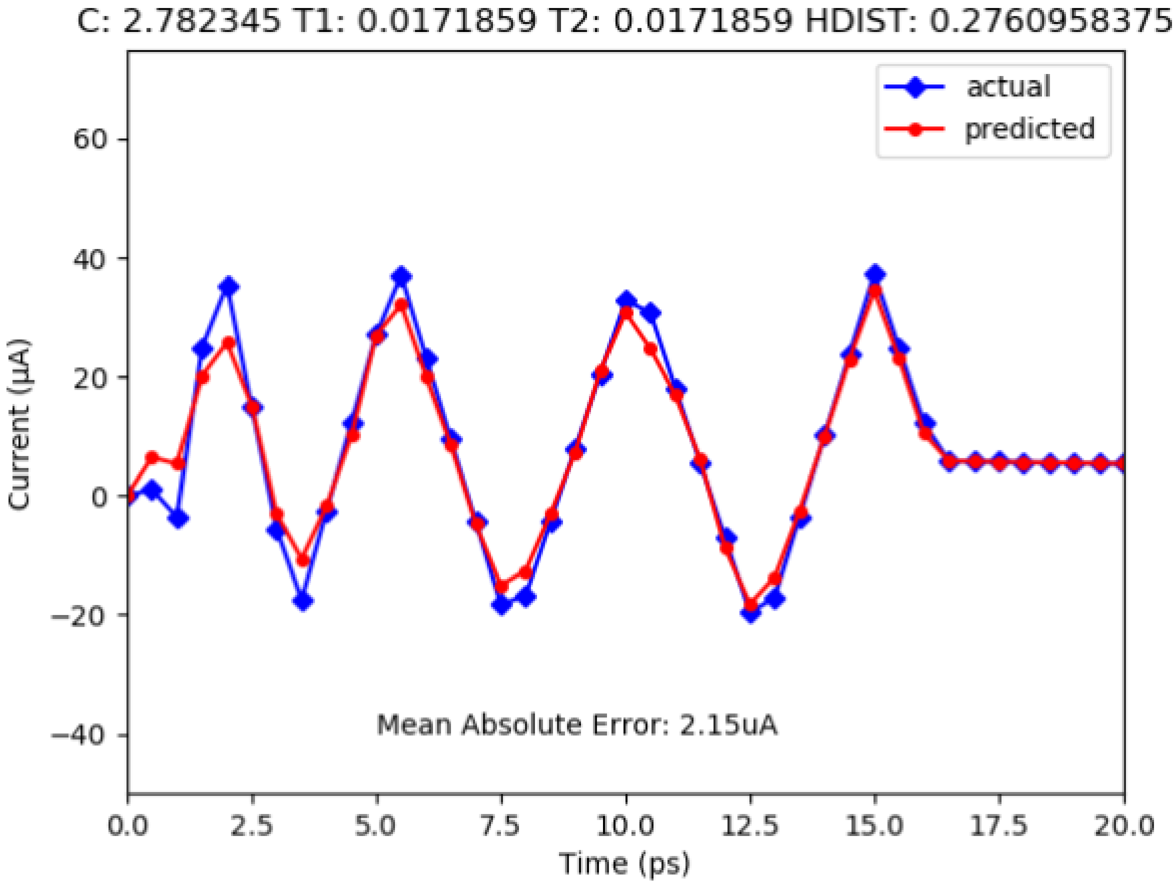

figure 22: An example of slight deviation (5.47% of cases)

Here there is a slight drop in accuracy, particularly apparent in peaks, where the predicted values are consistently closer to 0 than the actual, which have a wider range. An interesting point is at the start, where, unlike the rest of the graph, the predicted and the actual values are almost opposite.

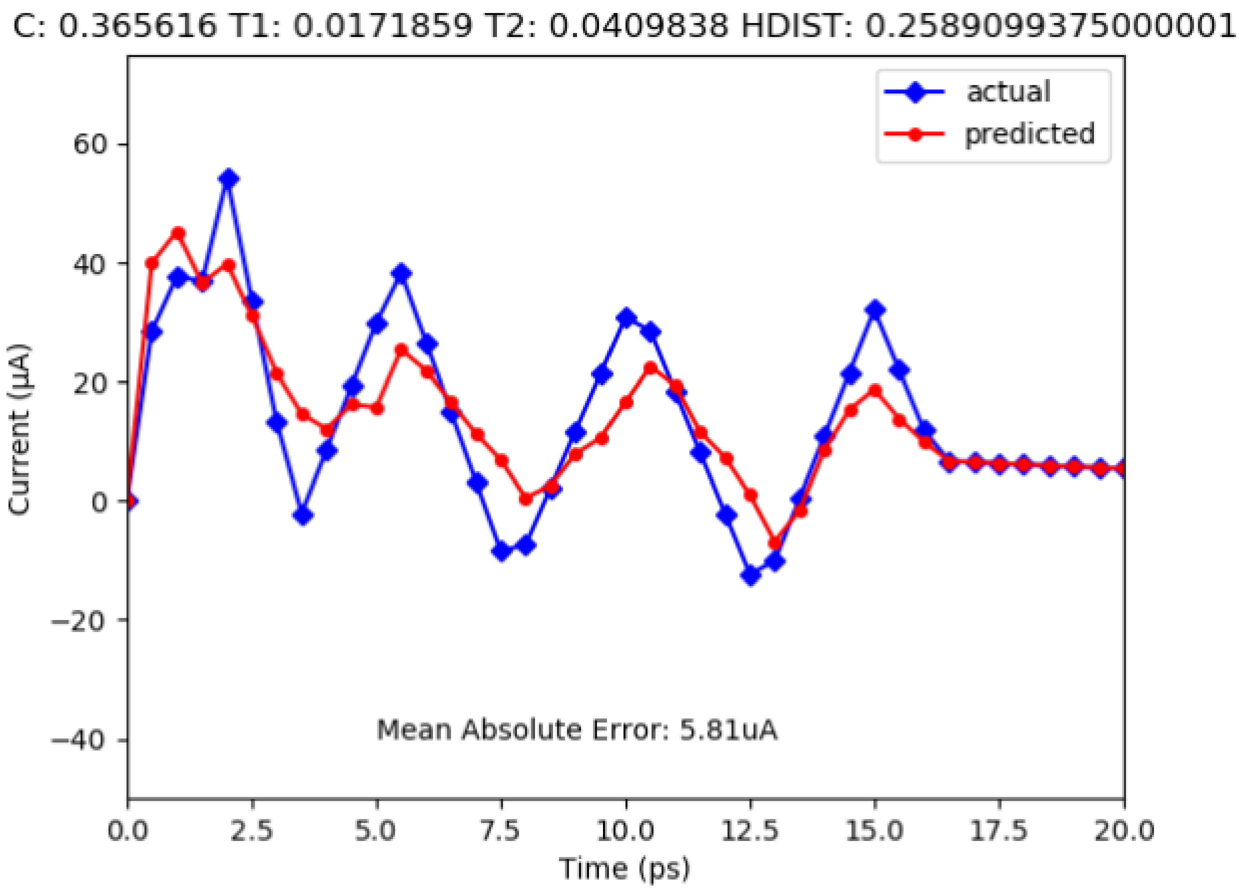

figure 23: An example of greater deviation (2.13% of cases)

In this case the accuracy is even lower. A saving grace however is that the pattern of peaks is more or less the same in both actual and predicted values. At 2.13% of cases, this is an expected and acceptable result of a machine learning approach.

Chapter 6

CONCLUSION AND FUTURE WORK

The purpose of this thesis was to examine whether machine learning could be used in glitch analysis, in order to approximate some circuit values during the unpredictable power dissipation caused by a glitch.

To this effect, using an HSPICE simulation of a simple two-input 45 nm NAND gate circuit, a dataset of multiple glitch cases was created. This dataset was then analyzed by two machine learning models based on random forest regression, one model predicting the output voltage and the other the power supply current.

The results show a reasonable accuracy of predictions, with average mean squared error of 0.00000003 for the output voltage and 5.17 for the power supply current. The average mean error was 0.1mV (0.001%) for the output voltage and 0.69 μA (15.64%) for the power supply current.

Even though the outcome seems promising, the scope was limited, so further work is needed to consolidate the efficiency of this preliminary research. Based on this thesis, some proposed next steps are the creation of models for predicting the behaviour exhibited by the output capacitor and grounding currents, as well as experimentation using different algorithms, datasets, logic gates, and advanced technology nodes (below 22nm).

Bibliography

[1] K. Chung, T. Kim and C. L. Liu, “A complete model for glitch analysis in logic circuits”, *Journal of Circuits, Systems, and Computers*, vol 11, No. 2, pp. 137-153, Feb. 2002

[2] R. Shah, “Glitch analysis and reduction in digital circuits”, *International Journal of VLSI design & Communication Systems (VLSICS)*, vol. 7, No. 4, pp. 47-55, Aug. 2016. doi: 10.5121/vlsic.2016.7405

[3] “CMOS, the Ideal Logic Family”, Fairchild Semiconductor Application Note 77, Jan. 1983

[4] J. Knight, “Glitches and Hazards in Digital Circuits”, Electronics Department - Carleton University, Mar. 2001

[5] T. M. Mitchell, *Machine Learning*, New York City NY: McGraw-Hill Science/Engineering/Math, Mar. 1997, p. 3

[6] E. Roberts, “History: The 1940’s to the 1970’s”, *Neural Networks - Sophomore College 2000*, Stanford University, 2000. Accessed on: July 22, 2020. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

[7] “History of Machine Learning”, 2018. Accessed on: July 22, 2020. [Online]. Available: https://www.doc.ic.ac.uk/~jce317/history-machine-learning.html

[8] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, vol 44, pp. 206–226, 1959. CiteSeerX 10.1.1.368.2254. doi:10.1147/rd.441.0206

[9] O. Child, (13 March 2016). "Menace: The Machine Educable Noughts And Crosses Engine", *Chalkdust Magazine*, 13 Mar, 2016. Accessed on: July 23, 2020. [Online]. Available: http://chalkdustmagazine.com/features/menace-machine-educable-noughts-crosses-engine

[10] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks" (PDF), May 2017. doi:10.1145/3065386

[11] C. Metz, *Google's AI Wins First Game in Historic Match With Go World Champion*, Wired, Sept. 3, 2016. Accessed on: July 23, 2020. [Online]. Available: https://www.wired.com/2016/03/googles-ai-wins-first-game-historic-match-go-champion/

[12] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York, NY: Springer Science+Business Media, 2006

[13] https://www.geeksforgeeks.org/decision-tree/

[14] Y. Lin, Y. Jeon, "Random Forests and Adaptive Nearest Neighbors", *Journal of the American Statistical Association*, 2002. CiteSeerX 10.1.1.153.9168

[15] "*HSPICE*", Accessed on July 24, 2020. [Online]. Available: https://www.synopsys.com/verification/ams-verification/hspice.html

[16] A. O. Troumpoulou, "Glitch Analysis", M. S. thesis, School of Eng., Univ. of Thessaly, Volos Greece, Oct. 2018. Accessed on Mar. 2020. [Online]. Available (in greek): https://www.e-ce.uth.gr/wp-content/uploads/formidable/59/Troumpoulou_angeliki-olympia.pdf

[17] W. McKinney, “Data Structures for Statistical Computing in Python”, *Proceedings of the 9th Python in Science Conference*, pp. 56-61, 2010. doi: 10.25080/Majora-92bf1922-00a

[18] F. Pedregosa et al, “Scikit-learn: Machine Learning in Python”, *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011

[19] J. D. Hunter, “Matplotlib: A 2D graphics environment”, *Computer in science & Engineering*, vol. 9-3, pp. 90-95, 2007. doi: 10.1109/MCSE.2007.55

[20] G. Van Rossum, “The Python Library Reference, release 3.7.6”, *Python Software Foundation*, 2020