Data Driven Techniques in Transformer Design

Aarush Goradia

13th December 2024

First Reader/Advisor: Professor Kaushik Sengupta

Second Reader: Professor Minjie Chen

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Electrical and Computer Engineering

Princeton University

I hereby declare that this Independent Work report represents my own work in accordance with university regulations.

Aarush Goradia

Data Driven Methods of Transformer Design

Aarush Goradia

When designing on-chip transformers, engineers typically begin by estimating the desired inductance and creating an approximate geometry using software like Cadence Virtuoso. This is followed by iterative simulations to validate the design and adjustments until the simulated inductance aligns with the target value. This manual process can be time-consuming and inefficient. In this paper, we propose an optimized design methodology utilizing a neural network trained on simulation data from randomly generated transformer geometries. By leveraging this approach, engineers can simply input their desired inductance value, and the neural network predicts the optimal geometry for achieving it. This significantly streamlines the design process, reduces trial-and-error iterations, and enhances efficiency in transformer development. Results demonstrate the potential of this method to revolutionize transformer design by providing precise and reliable geometry predictions tailored to specific requirements.

Acknowledgements

Thank you to Emir Karahan and Professor Sengupta for all your support and guidance with this project.

Table of Contents

1. Introduction

The design of on-chip transformers is a critical aspect of modern integrated circuit (IC) development, particularly for applications in RF communication, power management, and signal processing. These components play a vital role in impedance matching, energy transfer, and signal isolation, but their design process is often labor-intensive and iterative. Engineers traditionally rely on tools like Cadence Virtuoso to model the geometry of a transformer, simulate its performance, and iteratively refine the design until the desired parameter is achieved, which can take hours for a single transformer. While effective, this process is time-consuming and requires tedious work. With the increasing demand for high-performance transformers in compact form factors, there is a growing need for more efficient and automated design methodologies. This paper introduces a neural network-based approach that leverages simulation data to predict the optimal geometry for achieving a desired parameter, streamlining the design process and reducing manual effort.

This form of design, called top-down design, is an alternate form of electromagnetic structure design that is being explored by Professor Kaushik Sengupta and the Integrated Microsystems Research Lab (IMRL) at Princeton University [1]. The recent work of Sengupta's lab has been researching the efficacy of this method of design when designing an impedance matching network, a system that is used for power amplifiers (PA). While their research works with only these impedance matching networks, this form of top-down design can also be used when designing transformers, which are becoming more complicated, making bottom-down design significantly more inefficient. Therefore, this paper expands upon the work done by the IMRL lab and takes this novel approach and to design transformers according to pre-set operational parameters.

Currently, when synthesizing transformers for their chips, designers follow a design methodology that starts with the lumped parameters (e.g., primary inductance L_p, secondary inductance L_s, coupling coefficient K, and parasitic capacitances) follows:

1. They use initial simulations to optimize lumped parameters for a specific use case.
2. Translate lumped model results into physical dimensions.
3. Combine electromagnetic simulation results with PA circuit models to validate and fine-tune the design [2].

The goal of this project is to speed up the second step by using a neural network written in Python to immediately give a set of geometries based on the lumped parameters. The data for training the neural network is taken from Cadence EMX, a simulation software that gives the lumped parameters for a specific geometric configuration. This project will specifically look at 2 coil systems where the first coil has an additional spiral in the middle, as shown in the figure below.

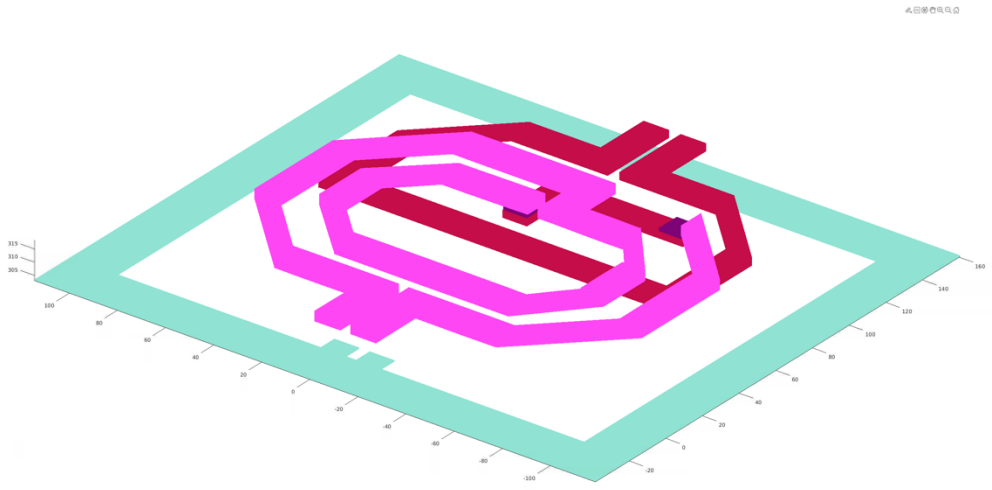

Figure 1: Bridged Transformer

2. Background

In traditional mm-wave InP circuits, microstrip transmission lines are predominantly used for power-combining and matching because they are easier to model across a larger range of frequencies and have easier design flows [2]. Conversely, though, these designs exhibit limitations such as large chip area and increased losses, which become more pronounced at higher frequencies. For this reason, transformer-based designs offer a promising alternative by addressing these constraints ad also enabling compact, broadband operation, and efficient harmonic control [3]. “A transformer usually exhibits these properties: (1) low series resistance in the primary and secondary windings, (2) high magnetic coupling between the primary and the secondary, (3) low capacitive coupling between the primary and the secondary, and (4) low parasitic capacitances to the substrate. Some of the trade-offs are thus similar to those of inductors” [3].

Transformers themselves operate on the principle of electromagnetic induction, where a varying current in the primary coil, which both coils are inductors, generates a magnetic field that induces a current in the secondary coil. This magnetic coupling allows for efficient energy transfer between the coils. The key parameters that determine the performance of a transformer, one that is part of the lumped parameters to be optimized are the primary (L_p) and secondary inductance (L_s), which is the tendency of the conductor to oppose a change. The efficiency of the energy transfer is quantified by the coupling coefficient, which follows this formula [2]:

$$K = \frac{M}{\sqrt{L_p \cdot L_s}}, 0 \leq K \leq 1 \qquad (1)$$

where M is the mutual inductance. In the case of on-chip transformers, a high K is desirable but limited by leakage inductance (which affects M), physical constraints, and high-frequency effects.

The efficiency of transformers is affected by several loss mechanics. To start, all inductors and transformers suffer from parasitic capacitance. This capacitance arises from the proximity of the transformer's metal traces and their interactions with the substrates. Unfortunately, in RF design, parasitic capacitance is unavoidable, so circuits have to be designed in order to reduce the effect of the parasitic capacitance to the absolute minimum. This capacitance also introduces significant limitations. Parasitic capacitance contribute to the self-resonance frequency (f_{SR}), at which the transformer stops acting like an inductor and starts acting like a capacitance, this is the absolute upper limit of frequency for a transformer on a chip [3]. Beyond the self-resonance frequency, capacitors are no longer efficient for energy transfer or impedance matching. Designers strive to minimize parasitic effects to push f_{SR} well beyond the intended operating range. The formula for f_{SR} is:

$$f_{SR} = \frac{1}{2\pi\sqrt{L_P C_P}} \qquad (2)$$

where C_P is parasitic capacitance. Apart from self-resonance, high parasitic capacitance also reduces the common mode rejection rate (CMRR), critical for differential circuits, like differential power amplifiers [3]. This measures the ability for a circuit to reject signals that are the same on both inputs while amplifying signals that are different on both inputs. Maintaining a high CMRR is important in RF design since it ensures proper functioning of the differential components and minimizes imbalances caused by parasitics or layout imperfections.

Transformers also have many forms of loss mechanisms that affect their efficiency on a chip. Transformers have a lumped parameter, Q, the quality factor, defined as "a measure of how much energy is lost in an inductor when it carries a sinusoidal current" [3]. Since only resistive components are subject to energy loss in a circuit, Q measures the resistances that form on the transformer leading to energy loss. If we suppose the metal line forming an inductor has series

resistance (R_S), the Q can then be defined as the ratio of desirable impedance over undesirable inductance:

$$Q = \frac{\omega_0 L}{R_S} \quad (3)$$

Unfortunately, the Q is not just dependent on the series resistance of the metal line, but also several other mechanisms that lower the Q further. Therefore, designers hope to design the inductor in a way where the resistance provides a Q factor of twice the desired value, assuming that other factors halve the Q factor after [3]. Moreover, at high frequencies, current through a conductor prefers to flow near the surface. Since it now flows through a smaller cross-sectional area, the resistance the current faces are higher.

Another problem that these transformers face is coupling to the substrate, whether it be capacitive or magnetic. Capacitive coupling happens because of the parasitic capacitance between the spirals and the substrate. As the voltage changes in the spiral, a displacement current flows through this capacitance and the substrate. Since the substrate has some finite resistivity, this causes loss in each cycle, lowering Q [3]. In magnetic coupling, a fraction of the magnetic flux generated by the inductor penetrates the substrate. These magnetic fields induce eddy currents in the substrate, leading to resistive losses. In this case, energy is lost as heat from the substrate, lowering the efficiency of the transformer and at higher frequencies, magnetic coupling losses become more significant due to the increased rate of change of magnetic fields.

One important use case of these on-chip transformers is in differential and power amplifying. With the rising want for more 5G communications, power amplifiers need innovation. Their primary role is to amplify weak RF signals to levels suitable for transmission while maintaining signal integrity. The amplified signals need to drive antennas effectively, particularly in phased-array systems that require precise amplitude and phase control. For mm-wave 5G applications,

where high-frequency bands like 28 GHz and 39 GHz are utilized, PAs must: provide sufficient gain and power output, maintain high efficiency to reduce power loss, and operate over broad or multiple frequency bands to support different network bands [4]. While transformers are “non-deal due to their finite coil inductance, high magnetic reluctance, and nonnegligible parasitics in the integrated circuit process”, by properly designing a transformer it is possible to emulate the behavior of higher order matching networks [2].

3. Methodology

This project will contain two main steps. It will first start with extending prior written code that generates random transformer geometries in Cadence Virtuoso. This prior code generates random geometries for two types of transformers: stacked and interleaved, as shown below.

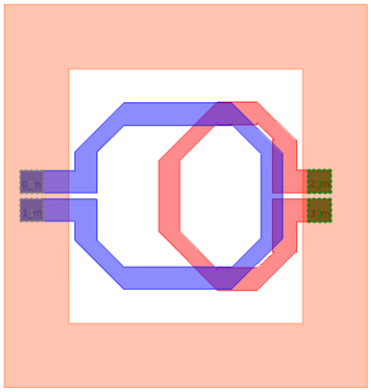

Figure 2: Stacked Transformer

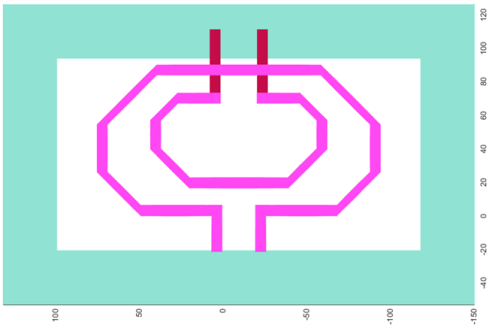

Figure 3: Interleaved Transformer

Therefore, the first goal will be to extend this code to randomly generate a bridged transformer, which is shown in Figure 1 above. This transformer has an inner coil for the primary coil that is connected by a bridge. An important difference in this design is making sure that the bridges and parts of the coil on the primary coil that are on the same layer as the secondary coil do not overlap at all, since that will cause a short circuit. The next part of this step will be to use a Python script to simulate the randomly generated geometry for some lumped parameters that will be described below. This uses Cadence EMX, which is a tool that will simulate the geometry and output the desired lumped parameters. The simulations for this project were all done at 20 GHz.

The second step of the project will be using that data to create a neural network that can predict the lumped parameters of a transformer solely based on the geometry that it is given. For this step, the EMX data is carefully manipulated into a table with 48 features and 5 labels. This is

data is run through a regression model with 3 layers and 64 neurons per layer. Initially, a ReLu activation process was used for the data, but upon further exploration, the Leaky ReLu activation seemed to fit the data better, giving better accuracy. The model also uses a 10-1-1 split for training, validation, and testing data. A diagram of the process is shown below:

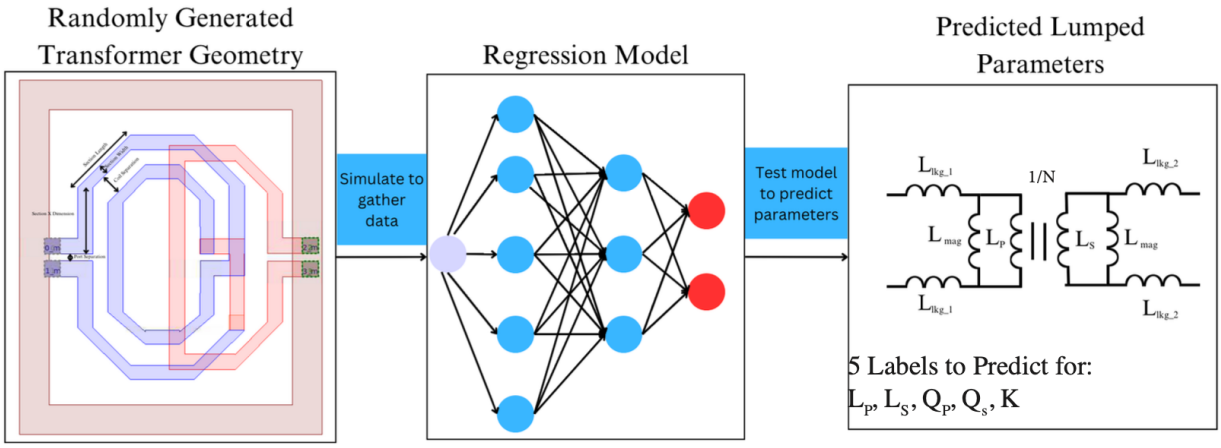

Figure 4: Neural Network Design Methodology

This project will optimize the transformers for five lumped parameters, L_P, L_S, Q_P, Q_S, and K. The formulas for these parameters are shown as below:

$$L_P = L_{mag} + L_{lkp}$$

$$L_s = \frac{L_P}{T^2}$$

$$Q_P = \frac{\omega_0 L_P}{R_P}$$

$$Q_S = \frac{\omega_0 L_S}{R_S}$$

$$K = \frac{M}{\sqrt{L_p \cdot L_s}}, 0 \leq K \leq 1$$

These parameters are the ones that the neural network will try to predict.

4. Results

The first step of the methodology was successful. Inner coils were made using certain geometries from their outer counterparts, with a set coil separation (this was varied when simulating data). The separation itself was the distance from a diagonal coil to its counterpart diagonal in the inner coil as shown by the black line:

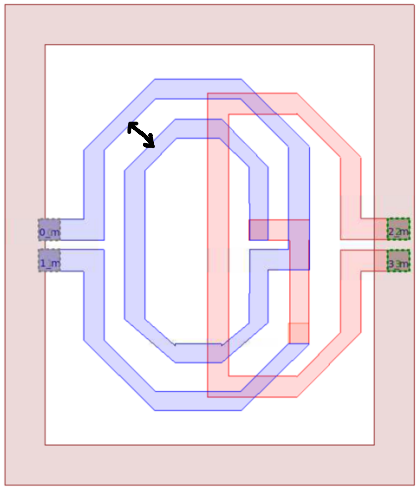

Figure 5: Bridged Transformer

The bridges were made with squares connecting the LD and OL layers through the VVBAR layer. While the generation of these polygons were successful, there were sometimes minor errors in the generation where certain polygons did not completely line up, or the geometry was slightly odd. This may have been due there sometimes being a mismatch of mathematical calculations for plotting these polygons.

Additionally, a method had to be created to make sure that the polygons of the second coil did not overlap with any polygons from the first coil. This was also successful but made it so that the simulation time was slightly slower due to it have to re-generate the polygons several times. Following this, the simulations were run.

The results for the training and testing were then plotted in Python and are as follows. The plotted results for the interleaved are as shown below, with 5000 points of data. The plot compares the predicted value with the actual value, and the accuracy is the mean percentage difference between all the points' predicted and actual values.

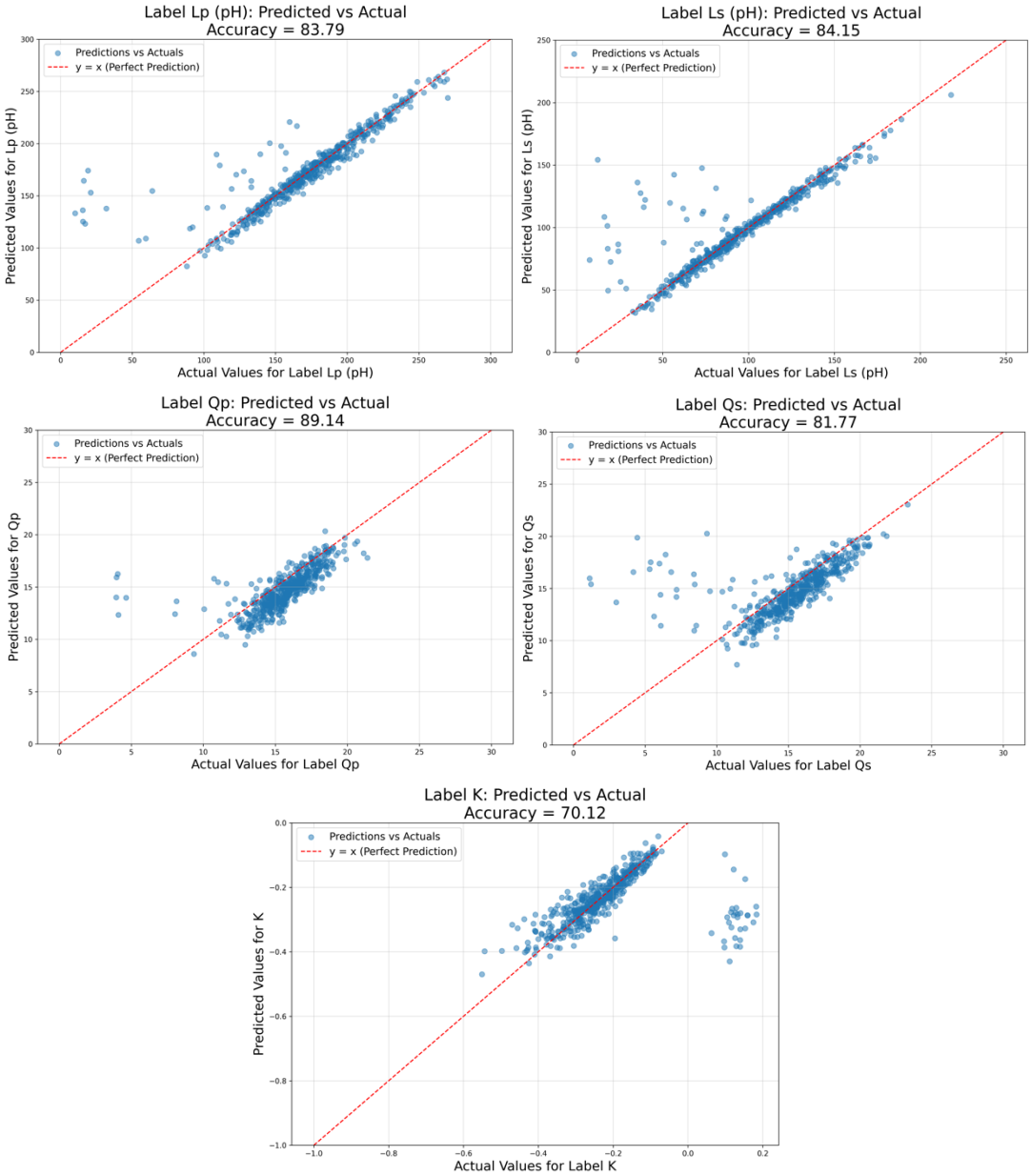

Figure 6: Interleaved Transformer Results

The same plots are shown below in Figure 7 for the stacked transformer, with ~6000 pieces of data. It is worth noting that the accuracy for all lumped parameters in the stacked model seem to be significantly higher; reasons for this will be discussed below.

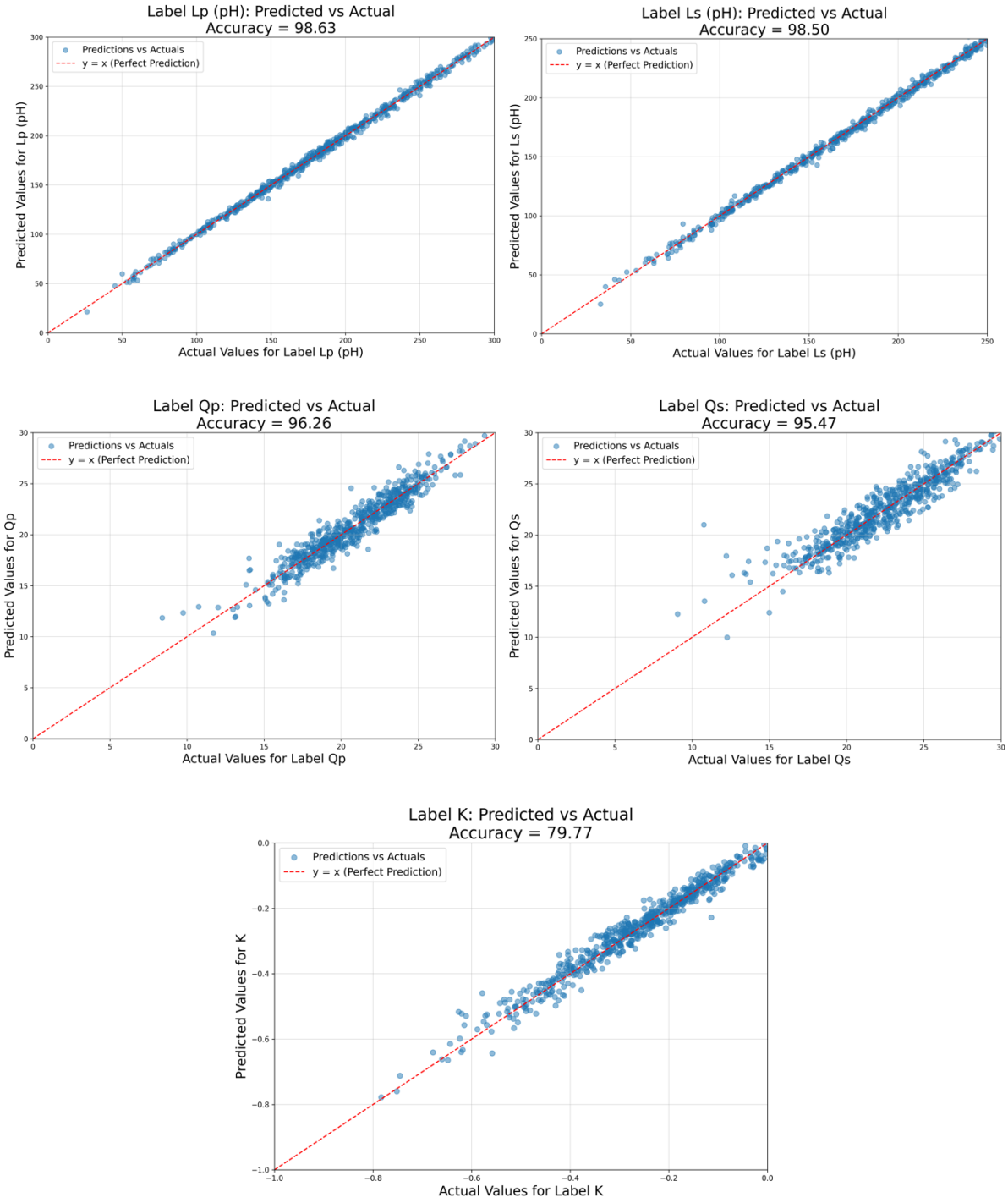

Figure 7: Stacked Transformer Results

The last set of results are for the bridged transformer. Again, the significance of the results will be discussed below.

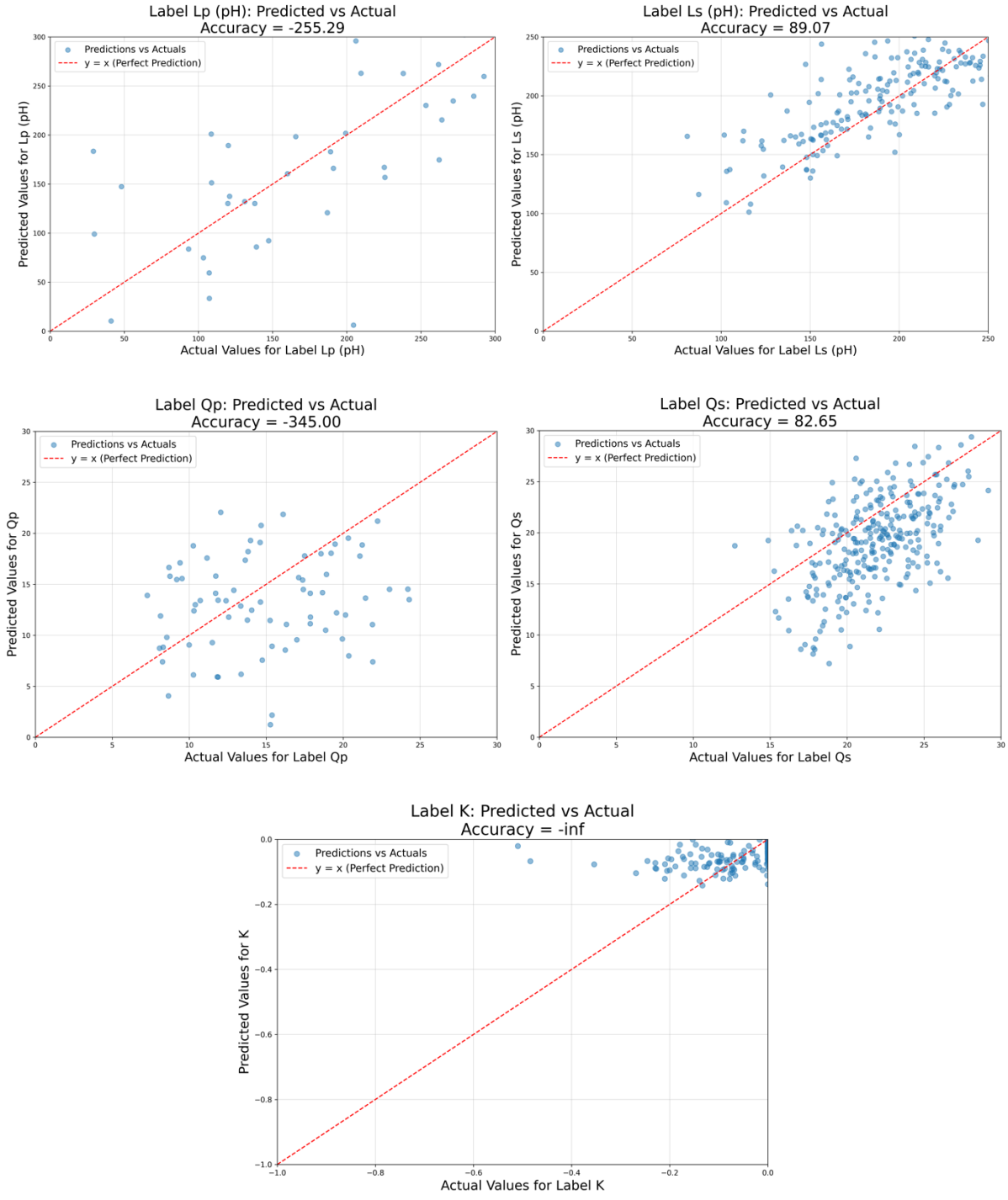

Figure 8: Results for Bridged Transformers

5. Analysis

From the simulation results shown above, it's obvious that the model is significantly more accurate when predicting what the lumped parameters would be for a stacked transformer than for an interleaved parameter. The most basic reasons for this could be that the model was better for that set of data, though this is unlikely. Moreover, maybe for the interleaved transformer, 5000 pieces of data was too few, which led to it not having the highest accuracy rate.

But a more promising answer to why the stacked transformer has a higher accuracy rate comes from examining the data that was put into the neural network. Below are two histograms, one for the data from the stacked transformer, and one from the interleaved.

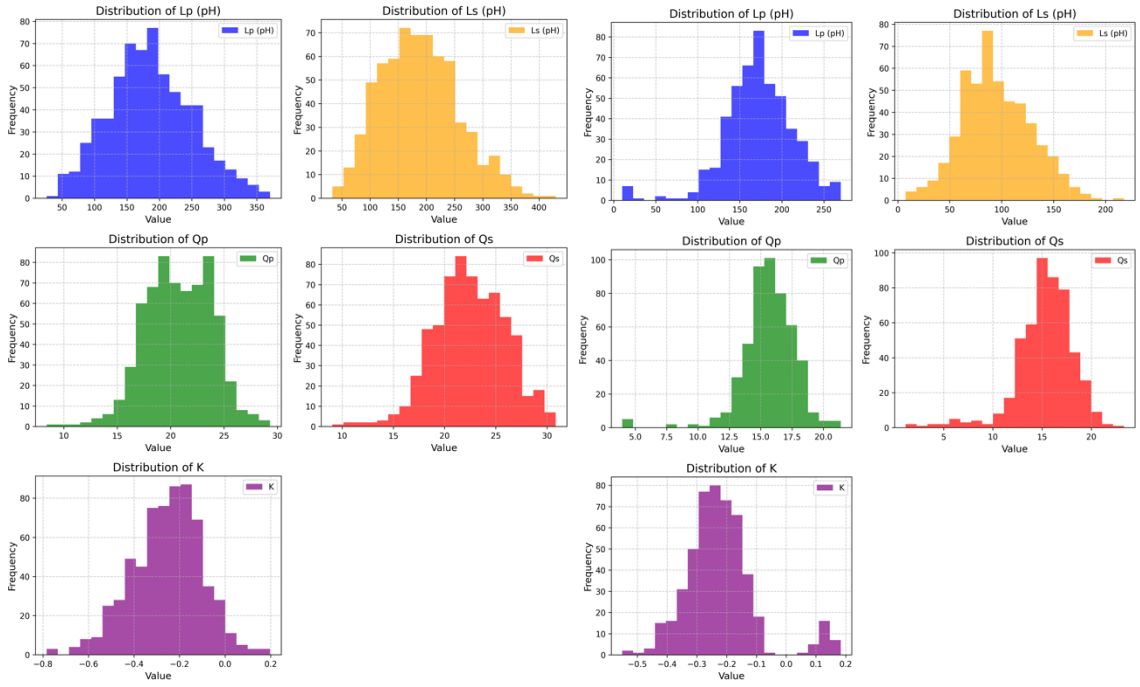

Figure 9: Distribution of Stacked Transformer

Figure 10: Distribution of Interleaved

It seems that the reason the accuracy was lower for the interleaved transformer was because the spread of data was lower. This is because almost always it is inside the first coil, hence there is a smaller area, and so the parameters will likely be skewed more towards one value. Therefore, since the data had a lower distribution, the model would have had a harder time predicting minority cases and cases there were further off that one main point. Moreover, in both cases, the *K* values were the worst predicted, and this may be due to the fact that there was some self-resonance in the data. This can be seen through the fact that there are data points for *K* that are larger than 0.

The bridged transformer seems to have results with extremely large error. This may be due to the generation of random polygons which themselves had some sources of error in polygons being lined up. Upon looking at the histogram, we can see that many points must have self-resonated since many have *K* values at/above 0. This error may be from a large amount of magnetic flux leakage from the geometry. It could also be a large amount of parasitic capacitance in the design. Lastly, it could also be a problem where the magnetic flux linkage is not maximized in the geometry. Which would be an error when scripting the transformers.

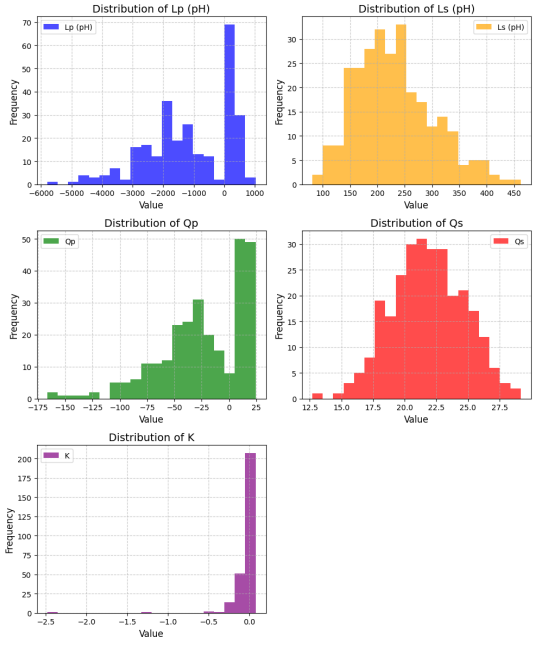

Figure 11: Distribution for Bridged Transformer

6. Discussion

The challenge in designing transformers is that the process is lengthy and tedious, with designers having to iteratively design these devices until their designs match their desired parameters. This requires a lot of grunt work for the designer in drawing out geometries, instead of focusing the designers' time onto something more important.

As shown, this method of transformer design strongly optimizes the lengthy and tedious process of iteratively designing a transformer. While the accuracy can be improved, this project shows promising results of a much more streamlined tool of RF electromagnetic structure synthesizing that could likely make the majority of all RF design significantly more efficient. Further work can also be done to create an end-to-end tool that is capable of synthesizing inductors and transformers on a chip for a given specification.

The overall accuracies for the prediction of lumped parameters were quite high for the stacked and interleaved transformers, which means this project was somewhat successful, but much more work can be done to ensure that the bridged transformers have better accuracy when predicting. For the bridged, this is likely due to the design, so more work can be put into the design of the randomly generated polygons such that they give more efficient, more capable transformers.

7. References

[1] E. A. Karahan, Z. Liu, and K. Sengupta, “Deep-Learning-Based Inverse-Designed Millimeter-Wave Passives and Power Amplifiers,” *IEEE J. Solid-State Circuits*, vol. 58, no. 11, pp. 3074–3088, Nov. 2023, doi: 10.1109/JSSC.2023.3276315.

[2] Z. Liu, T. Sharma, C. R. Chappidi, S. Venkatesh, Y. Yu, and K. Sengupta, “A 42–62 GHz Transformer-Based Broadband mm-Wave InP PA With Second-Harmonic Waveform Engineering and Enhanced Linearity,” *IEEE Trans. Microw. Theory Tech.*, vol. 69, no. 1, pp. 756–773, Jan. 2021, doi: 10.1109/TMTT.2020.3037092.

[3] B. Razavi, *RF microelectronics*, 2. edition. in The Prentice Hall communications engineering and emerging technologies series. Upper Saddle River, NJ Munich: Prentice Hall, 2012.

[4] M. A. Mokri, S. Miraslani, M. A. Hoque, and D. Heo, “A Dual-Path Transformer-Based Multiband Power Amplifier for mm-Wave 5G Applications,” *IEEE J. Solid-State Circuits*, vol. 59, no. 6, pp. 1643–1655, Jun. 2024, doi: 10.1109/JSSC.2023.3344073.

8. Appendix I: Inner Coil Code

```
pythag = np.sqrt((coil_separation**2)/2)

        point_1 = [-excess_centering+section_2_width+pythag, -port_seperation]
        point_2 = [-excess_centering+section_2_width+pythag+section_1_width, -port_seperation]
        #point_3 = [section_1_width+coil_separation,section_1_y_dim]
        #point_4 = [coil_separation+section_1_width,section_1_y_dim]
        point_3 = [-excess_centering+section_2_width+pythag+section_1_width, section_1_y_dim-pythag]
        point_4 = [-excess_centering+section_2_width+pythag, section_1_y_dim-pythag]
        poly1_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1, point_2, point_3, point_4 ])
        self.em_structure.append(poly1_inner)

        excess_centering = (section_2_width - section_1_width)/2

        point_1 = [-excess_centering+section_2_width+pythag, section_1_y_dim-pythag]
        point_2 = [-excess_centering+section_2_width+pythag+section_2_width, section_1_y_dim-pythag]
        point_3 = [-excess_centering+section_2_x_dim+pythag+section_2_width, section_1_y_dim+section_2_y_dim-section_3_width-pythag]
        point_4 = [-excess_centering+section_2_x_dim+pythag, section_1_y_dim+section_2_y_dim-section_3_width-pythag]
        poly2_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1, point_2, point_3, point_4 ])
        self.em_structure.append(poly2_inner)

        point_1 = [-excess_centering+section_2_x_dim+pythag, section_1_y_dim+section_2_y_dim-pythag-section_3_width]
        point_2 = [-excess_centering+section_2_x_dim+pythag, section_1_y_dim+section_2_y_dim-section_3_width-pythag-section_3_width]
        point_3 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag, section_1_y_dim+section_2_y_dim-section_3_width-pythag-section_3_width]
        point_4 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag, section_1_y_dim+section_2_y_dim-pythag-section_3_width]
        #ref_point = [-excess_centering+section_2_x_dim, section_1_y_dim+section_2_y_dim]
        poly3_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1, point_2, point_3, point_4 ])
        self.em_structure.append(poly3_inner)

        point_1 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag-section_4_width, section_1_y_dim+section_2_y_dim-section_3_width-pythag]
        point_2 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag, section_1_y_dim+section_2_y_dim-section_3_width-pythag]
        point_3 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag, section_1_y_dim+section_2_y_dim-section_4_y_dim-pythag]
        point_4 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-section_5_width, section_1_y_dim+section_2_y_dim-section_4_y_dim-pythag]
        #ref_point = [-excess_centering+section_2_x_dim, section_1_y_dim+section_2_y_dim]
        poly4_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1, point_2, point_3, point_4 ])
        self.em_structure.append(poly4_inner)

        excess_centering_2 = (section_4_width - section_5_width)/2
        point_1 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-section_5_width, section_1_y_dim+section_2_y_dim-section_4_y_dim-pythag]
        point_2 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag, section_1_y_dim+section_2_y_dim-section_4_y_dim-pythag]
        point_3 = [excess_centering_2-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,0]
```

```
41.         point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, 0]
42.         poly5_1_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ])
43.         self.em_structure.append(poly5_1_inner)
44.
45.         point_1 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width-pythag-section_5_width, section_1_y_dim+section_2_y_dim-section_4_y_dim-
section_5_y_dim+pythag]
46.         point_2 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width-pythag, section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+pythag]
47.         point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,-
port_seperation]
48.         point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, -port_seperation]
49.         poly5_2_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ])
50.         self.em_structure.append(poly5_2_inner)
51.
52.         point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+pythag]
53.         point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-
pythag,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+pythag]
54.         point_3 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag, -section_1_y_dim-
section_2_y_dim-port_seperation+coil_separation+section_3_width]
55.         point_4 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag-section_5_width, -
section_1_y_dim-section_2_y_dim-port_seperation+coil_separation+section_3_width]
56.         poly6_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ] )
57.         self.em_structure.append(poly6_inner)
58.
59.         point_1 = [-excess_centering+section_2_width+section_2_x_dim+pythag-section_2_width, -
section_1_y_dim-section_2_y_dim-port_seperation+coil_separation+section_3_width]
60.         point_2 = [-excess_centering+section_2_width+section_2_x_dim+pythag-section_2_width, -
section_1_y_dim-section_2_y_dim+section_3_width-port_seperation+coil_separation+section_3_width]
61.         point_3 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag, -section_1_y_dim-
section_2_y_dim+section_3_width-port_seperation+coil_separation+section_3_width]
62.         point_4 = [-excess_centering+section_2_x_dim+section_3_x_dim-pythag, -section_1_y_dim-
section_2_y_dim-port_seperation+coil_separation+section_3_width]
63.         poly8_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ] )
64.         self.em_structure.append(poly8_inner)
65.
66.         excess_centering = (section_2_width - section_1_width)/2
67.         point_1 = [-excess_centering+section_2_width+pythag, -section_1_y_dim-
port_seperation+pythag]
68.         point_2 = [-excess_centering+section_2_width+pythag+section_2_width, -section_1_y_dim-
port_seperation+pythag]
69.         point_3 = [-excess_centering+section_2_width+section_2_x_dim+pythag, -section_1_y_dim-
section_2_y_dim-port_seperation+coil_separation+section_3_width]
70.         point_4 = [-excess_centering+section_2_width+section_2_x_dim+pythag-section_2_width, -
section_1_y_dim-section_2_y_dim-port_seperation+coil_separation+section_3_width]
71.         poly9_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ])
72.         self.em_structure.append(poly9_inner)
73.
74.         excess_centering = (section_2_width_sec - section_1_width_sec)/2
75.         point_1 = [-excess_centering+section_2_width+pythag, -port_seperation]
76.         point_2 = [-excess_centering+section_2_width+pythag+section_1_width, -port_seperation]
```

```
77.         point_3 = [-excess_centering+section_2_width+pythag+section_2_width, -section_1_y_dim-
port_seperation+pythag]
78.         point_4 = [-excess_centering+section_2_width+pythag, -section_1_y_dim-
port_seperation+pythag]
79.         poly10_inner = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ] )
80.         self.em_structure.append(poly10_inner)
81.
82.         # bridging polygons
83.         # Adjust diagonal polygons for 45-degree angles
84.
85.         point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width,-port_seperation-section_1_width]
86.         point_4 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim, -
port_seperation-section_1_width]
87.         point_1 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim, -
port_seperation]
88.         point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width,-port_seperation]
89.         bridge_left_right = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [point_1,
point_2, point_3, point_4])
90.         self.em_structure.append(bridge_left_right)
91.
92.
93.         point_1 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width, -port_seperation-section_1_width]
94.         point_2 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim, -
port_seperation-section_1_width]
95.         point_3 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim,
section_1_y_dim+section_2_y_dim-section_4_y_dim]
96.         point_4 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width, section_1_y_dim+section_2_y_dim-section_4_y_dim]
97.         bridge_up_1 = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [point_1,
point_2, point_3, point_4])
98.         self.em_structure.append(bridge_up_1)
99.
100.        # second diagonal. Going from point on same line instead opposite. Excess centering
off.
101.        excess_centering = (section_2_width - section_1_width)/2
102.
103.        point_1 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width-pythag-section_5_width, 0]
104.        point_2 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width-pythag-section_5_width, section_1_width]
105.        point_3 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim,
section_1_width]
106.        point_4 = [-excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim, 0]
107.        bridge_right_left = self.ws.db.create_polygon(self.cv, [LAYER3, "drawing"], [point_1,
point_2, point_3, point_4])
108.        self.em_structure.append(bridge_right_left)
109.
110.        excess_centering_2 = (section_4_width - section_5_width)/2
111.        point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width, section_1_width]
112.        point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_width]
113.        point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim]
```

```
114.         point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim]
115.         poly5 = self.ws.db.create_polygon(self.cv, [LAYER3, "drawing"], [ point_1, point_2,
point_3, point_4 ])
116.         self.em_structure.append(poly5)
117.
118.         point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+section_1_width/2]
119.         point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+section_1_width/2]
120.         point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim-section_1_width/2]
121.         point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim-section_1_width/2]
122.         bridge_2_ld = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ])
123.         self.em_structure.append(bridge_2_ld)
124.
125.         point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+section_1_width/2]
126.         point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+section_1_width/2]
127.         point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim-section_1_width/2]
128.         point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim-section_1_width/2]
129.         bridge_2_ol = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ])
130.         self.em_structure.append(bridge_2_ol)
131.
132.         point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+section_1_width/2]
133.         point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim+section_1_width/2]
134.         point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width+section_5_width,
section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim-section_1_width/2]
135.         point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-
section_4_width,section_1_y_dim+section_2_y_dim-section_4_y_dim-section_5_y_dim-section_1_width/2]
136.         bridge_2_vvbar = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [ point_1,
point_2, point_3, point_4 ])
137.         self.em_structure.append(bridge_2_vvbar)
138.
139.
140.         point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,
section_1_width*(3/2)]
141.         point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, section_1_width*(3/2)]
```

```
142.          point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,
section_1_width/2]
143.          point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, section_1_width/2]
144.          bridge_1_ld = self.ws.db.create_polygon(self.cv, [LAYER2, "drawing"], [point_1,
point_2, point_3, point_4])
145.          self.em_structure.append(bridge_1_ld)
146.
147.          point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,
section_1_width*(3/2)]
148.          point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, section_1_width*(3/2)]
149.          point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,
section_1_width/2]
150.          point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, section_1_width/2]
151.          bridge_1_ol = self.ws.db.create_polygon(self.cv, [LAYER3, "drawing"], [point_1,
point_2, point_3, point_4])
152.          self.em_structure.append(bridge_1_ol)
153.
154.          point_2 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,
section_1_width*(3/2)]
155.          point_1 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, section_1_width*(3/2)]
156.          point_3 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag,
section_1_width/2]
157.          point_4 = [excess_centering_2-
excess_centering+section_2_x_dim+section_3_x_dim+section_4_x_dim-section_4_width-pythag-
section_5_width, section_1_width/2]
158.          bridge_1_vvbar = self.ws.db.create_polygon(self.cv, [LAYER4, "drawing"], [point_1,
point_2, point_3, point_4])
159.          self.em_structure.append(bridge_1_vvbar)
```

9. Appendix II: Neural Network Training Code

```
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import LeakyReLU, Dense

data = scipy.io.loadmat('/content/combined_table.mat')['combined_table']
features = data[:, :48]
labels = data[:, 48:]
labels[:, 4] *= 100

# Split the data
X_train, X_temp, y_train, y_temp = train_test_split(features, labels,
test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

# Verify splits
print(f"Training set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
print(f"Testing set size: {X_test.shape[0]}")

# Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, input_shape=(48,)),
    LeakyReLU(alpha=0.1),
    tf.keras.layers.Dense(64),
    LeakyReLU(alpha=0.1),
    tf.keras.layers.Dense(64),
    LeakyReLU(alpha=0.1),
    tf.keras.layers.Dense(5)
])
# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    batch_size=32,
    verbose=2
)
```

```
# Evaluate the model on the test set
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=2)
print(f"Test Loss: {test_loss}")
print(f"Test MAE: {test_mae}")

import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```