MTP2_Gourav_final_version

by Gourav Tilwankar

Submission date: 30-Jun-2022 09:04PM (UTC+0530)
Submission ID: 1865051403
File name: MTP_PHASE_2_Gourav_vfinal_1.pdf (1.5M)
Word count: 9285
Character count: 44903

Abstract

Humans create machines to reduce time and effort, Machines evolve to learn and work effectively, and humans work to improve machines. Hence, applying Machine Learning to create those machines can reduce much effort. In the present time, creating Analog Circuits using ML techniques can reduce designers' much effort. The task here is to create one circuit design and ask for output specifications of the particular circuit from the user to give the circuit input parameters using ML techniques. Also, using the same techniques to include process variation in circuits. This methodology includes proper circuit design for one specification, collecting datasets using software simulations, and predicting output using machine learning models and processed datasets. We chose the research work because we rarely see a combination of Analog Circuits and ML in this area. Once completed, it can almost revolutionize the way of circuit designing in today's world.

In this work, we designed the circuit in Cadence Virtuoso software. We obtain comma-separated value (.csv) files as a dataset containing the input-output behavior of the circuit, and Machine Learning code is written in Python using the Support Vector Regression methodology. A final result is a tool that takes the specification of a particular circuit from the user and provides the circuit sizes as output which can be fed manually to Cadence Software for output verification. This work closes the loop of designing the circuits with machines faster and reduces the designer's effort.

Chapter 1

Introduction

1.1 Background

Intelligent machines are driving today's world, which works on machine learning, from handy Mobile Phones to large industrial machines to space satellites and various other applications. All of these human-made creations include circuits that help in the proper execution of Artificial Intelligence for multiple applications. These circuits are the result of years of hard work by researchers across the globe. We are designing systems to run Machine Learning models. In that case, we can also create machine learning models to develop these systems and circuits, which can help the circuit designer achieve their goals quickly and efficiently. [7] gives an idea about areas where we can do automation with the help of artificial intelligence in analog circuit design. Also, the currently used technology processes have changed a lot in the past few decades. We have moved from one technology to another to create compact circuits in digital and analog fields. One significant limitation in this conversion is systems and circuits need to be developed from scratch again. In this research work, we are designing and developing machine learning models which can do this conversion process accurately to some extent. It can help the designer start with an early known circuit analysis, reducing design time.

1.2 Motivation

Regarding the topic of Machine Learning in Analog Circuits, there have been studies in the past few decades which tell that Machine Learning has influenced this field. Studies and findings are

going on by Scientists and Researchers across the globe to apply Machine Learning in Analog Circuits. Some are Automated design, Optimized Chip design, Placement and Routing, Fault Diagnosis, and many others. [1] provides information about individual parameters of specific circuits with the help of an Optimized dataset. It is just an example of where ML algorithms can be fruitful in designing Analog Circuits. Mostly, the studies [4], [5], [8] show the result on the same circuit for either optimized parameters for a given specification or outputs of circuits for given parameters. There has been an unexplored area where we can use Machine Learning algorithms. It is to change the technology process from one to another to make the designing aspect much more effortless.

1.3 Project Objective

The project aims to create a Machine Learning tool that takes the specifications of a particular circuit. It uses a pre-generated dataset of the same circuit containing the relationship between inputs variables and output specifications obtained from the Cadence Virtuoso environment as a comma-separated value (.csv) file. At last, it predicts the values of input variables to be fed by the user to Cadence Software to get a pre-designed circuit netlist with near or equivalent specifications as required. These pre-computed specifications and designs will reduce the overall time of designing the analog circuit and use it for various purposes.

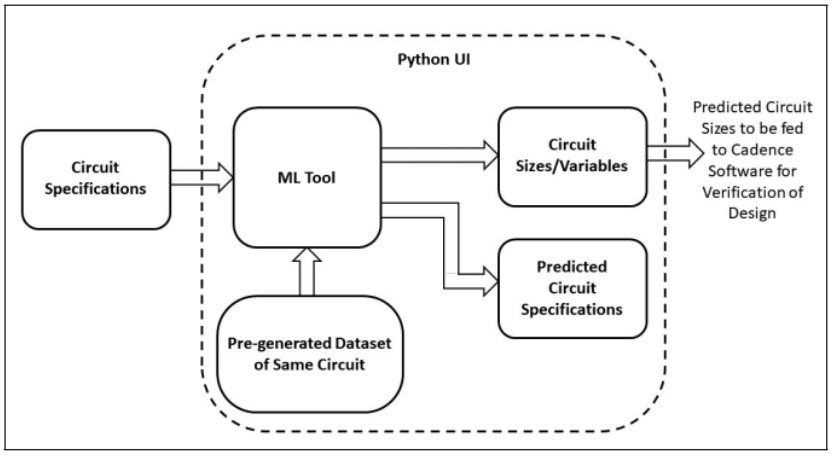

Figure 1.1: Block Diagram of Project Objective

1.4 Thesis outline

The subject matter of the thesis is presented in the following ten chapters.

- ✓ Chapter 1 introduces the project topic and briefly describes why we chose this project.
- ✓ Chapter 2 overviews the work done during MTP Phase 1 and all our problems. The methodology and steps that we took are briefly described. The circuits chosen for the project and the machine learning model selected are explained.
- ✓ Chapter 3 describes the Circuit Selection strategies followed in Phase 2 of the project. Selecting a circuit and modifying it for better functionality while ensuring the project's aim can be fulfilled. It briefly explains how the final circuit netlist is selected for further project stages.
- ✓ Chapter 4 describes the challenges faced in designing the circuit. The problem encountered while converting Circuit parameters into Machine Learning compatible variables is summarized here.
- ✓ Chapter 5 provides information about the steps to generate datasets from the Cadence Virtuoso environment. Information about variables, multiple datasets obtained, and Final Dataset preparation for later project stages is given here.
- ✓ Chapter 6 describes the Machine Learning part of the project. It gives information about the techniques used for data processing, creating accurate models, and minimizing the errors in the prediction algorithm.
- ✓ Chapter 7 summarizes the various methods of testing the model created. It describes how the tool works and what information the user will gain from working with this tool usage.
- ✓ Chapter 8 concludes the results and a summary of methods, techniques, and algorithms formulated for the duration of this project.

Chapter 2

Overview of MTP Phase 1

This chapter overviews the work done in EE797 MTP Phase-I. During phase 1, we implemented the project idea from scratch. As the project title suggests, the main keywords of the project are Analog Circuit Design and Machine Learning. Hence, we divided the main concentration into three steps:

1. Circuit Selection
2. Dataset Generation
3. Machine Learning Model

2.1 Circuit Selection

The most critical aspect of starting the project was selecting a circuit with various implementations/applications in Analog Circuit Design. This circuit needs to be designed by different users for various purposes. Also, the circuit needs inputs and outputs so that the machine learning aspect can be applied and used fruitfully. Therefore, the approach was to gradually start from the most basic circuit and upgrade in circuit complexity. The circuits chosen were as follows:

1. NMOS
2. CMOS Inverter
3. 2-stage Operation Amplifier

For every circuit, we followed various circuit design aspects needed in Cadence Software like Netlist Creation, Testbench Creation, and Simulation. Before extracting data for further process, it was required to carefully select the input and output variables needed for a circuit. The entire process, from creating netlist to data extraction, was done for the first two circuits, i.e., NMOS and CMOS Inverter. For 2-stage Op-Amp, we created only the netlist.

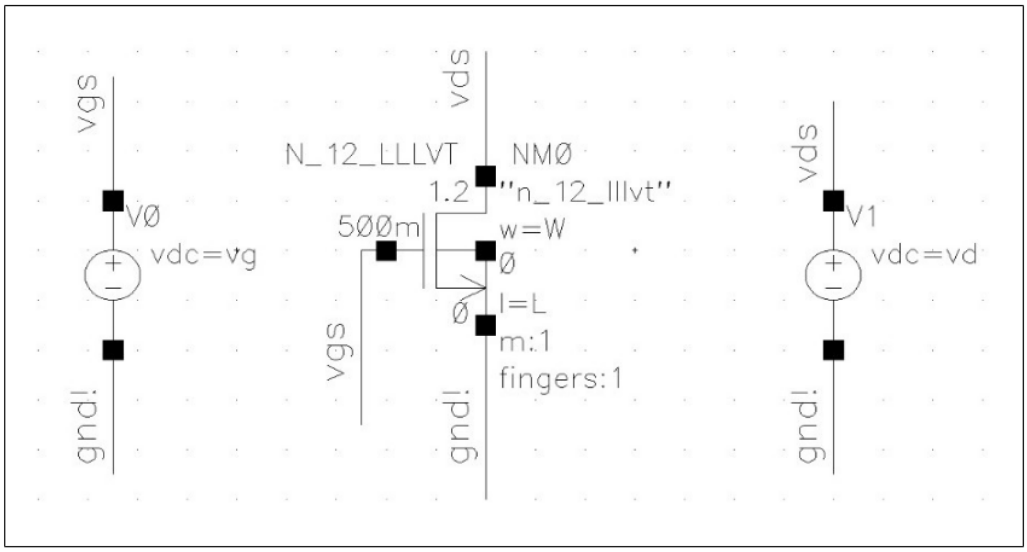

Figure 2.1: Netlist of NMOS

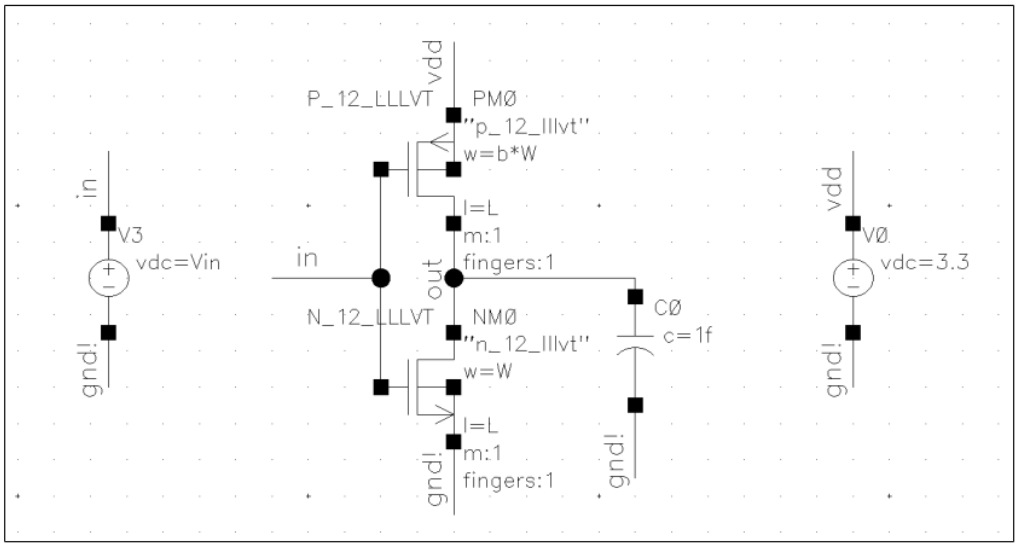

Figure 2.2: Netlist of CMOS Inverter

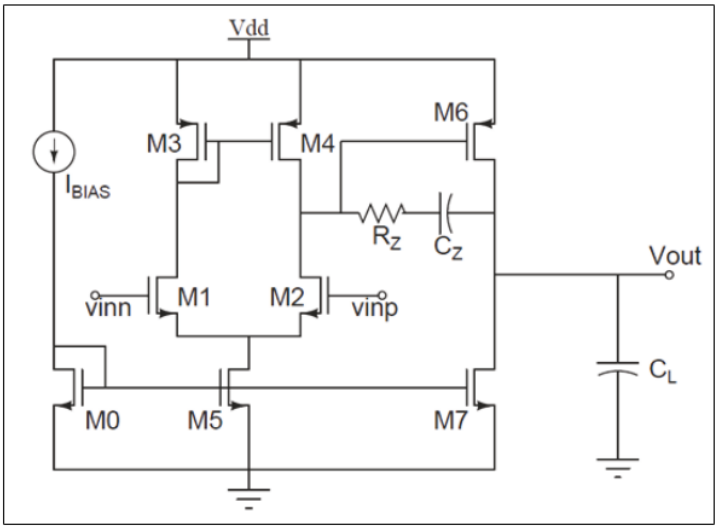

Figure 2.3: Schematic of 2-stage Op-Amp

2.2 Dataset Collection

The step of dataset generation depended on whether the simulations run in the Cadence Virtuoso environment were proper and valid enough. Also, the number of inputs and outputs is limited in the above-described circuits and, therefore, could not create a proper dataset. Due to these specific limitations, the data extracted was not of much relevance, and hence, the further process of applying the Machine Learning algorithm was not of use.

Table 2.1: Different Circuits used in MTP-1, their Inputs and Outputs

Circuits	Inputs	Outputs
NMOS	V_{GS}, V_{DS}, Transistor Width and Length	g_m, I_d, R_{ON}, Region of Operation, Early Voltage, Channel Length Modulation (λ), Threshold Voltage, Overdrive Voltage
CMOS Inverter	Input Voltage, Transistor Lengths, Widths	Output Voltage
2-stage Op-Amp	Transistor Widths, Lengths, Multipliers	DC Gain, Slew Rate, DC Power, Phase Margin

2.3 Machine Learning Model

Although the previous steps were incomplete, we had to be ready with an ML algorithm that, when we obtain the proper dataset, would be able to do the task of predicting output based on the input given. Since Machine Learning output was required to be of regression type, we applied the most basic version of ML technique, Linear Regression with Gradient Descent algorithm, for weight update of inputs.

We update the weight update after every iteration of training based on the Gradient Descent Rule. This rule considers that the error increases or decreases in a particular variation of the weight vector. The weight update takes place such that the error decreases the most. Finally, we use the weight matrix to test external input for prediction after reaching the number of iterations.

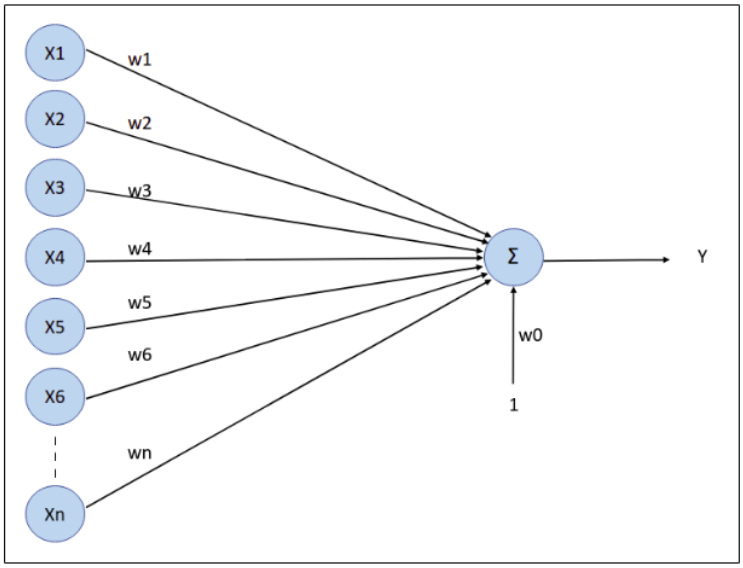

Figure 2.4: Linear Regression Technique

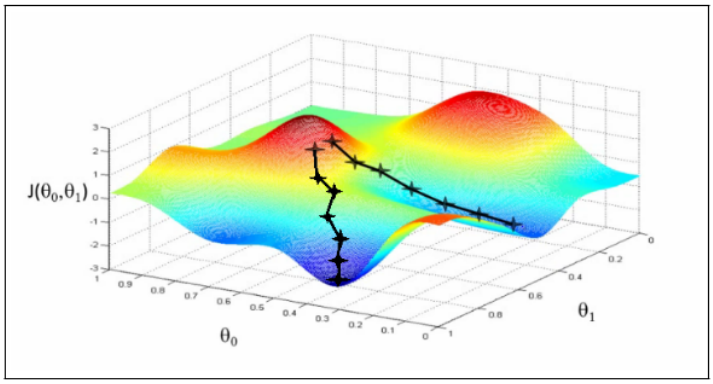

Figure 2.5: Gradient Descent Algorithm

Figure 2.5 shows gradient descent updates of loss function loss J for two input features θ_0 and θ_1. As explained earlier, this rule focuses on finding the direction of weight update where error gets decreased the most, trying to find local/global minima.

Chapter 3

Circuit Selection and Designing

After Project phase 1, we modified the objective by setting up the circuit selection and designing as the top priority. We put aside the part of Process variation in the circuit and focus on setting up a Machine Learning tool that does not do process variation but gives values of circuit parameters for given specifications based on provided dataset. Hence, a suitable circuit was required, which has some applications, and it takes time for analog circuit designers to create that circuit netlists, testbenches, and validate its working. This action will help us prove the motive for doing the project.

We selected a 2-stage single-ended Operational Amplifier with Miller Compensation as the final circuit, but the schematic/netlist wasn't confirmed/ finalized. It depended upon what all functionalities we wanted from the circuit. Hence, we thought from the user's perspective.

A user/designer wants basic terminologies like Gain, Bandwidth, Power, Slew Rate, etc., to be given as specifications and wants to control the transistor sizes with a minimized number of variables, most optimum relations between the parameters, and a maximum number of outputs.

The Circuit Netlist finalization took place in 3 steps.

3.1 Basic Circuit

The most common 2-stage Op-Amp with Miller Compensation was selected. Various state-of-the-art techniques on this circuit produce results ranging from low power, high gain, high slew rate, high bandwidth, or achieving specific specifications for particular applications. Figure 3.1 represents the netlist for the primary circuit with input variables to be determined yet and output

specifications as follows:

- DC Gain
- DC Power
- Slew Rate
- Unity Gain Bandwidth
- Noise
- Phase Margin

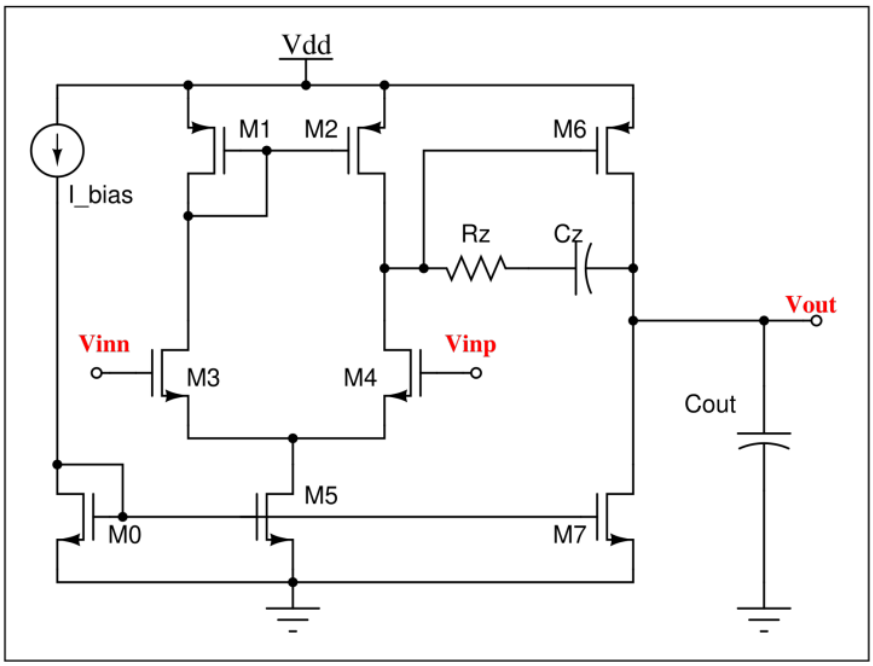

Figure 3.1: 2-Stage Single-Ended Operational Amplifier Netlist [5]

The circuit has limitations as we could not track the value of the resistor and the capacitor when there is temperature variation. Hence, we modified the circuit to a differential amplifier was taken for the study.

3.2 2-Stage Differential Amplifier Circuit

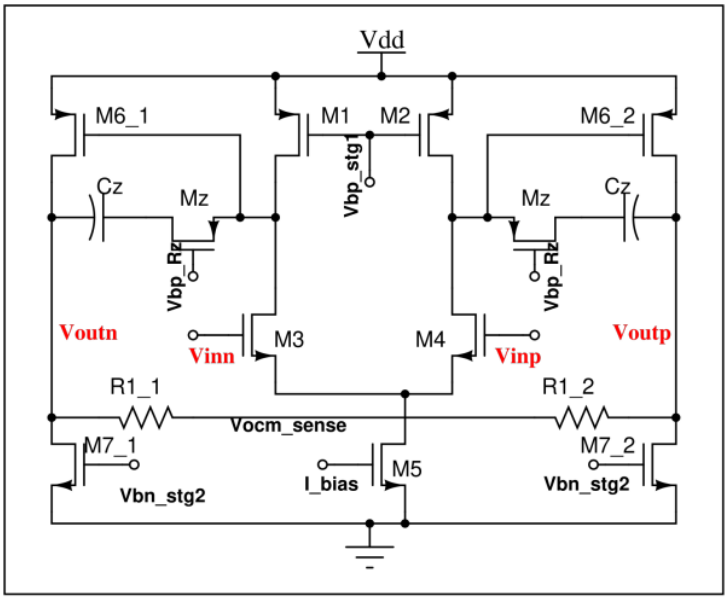

Figure 3.2: 2-Stage Differential Amplifier Netlist (a)

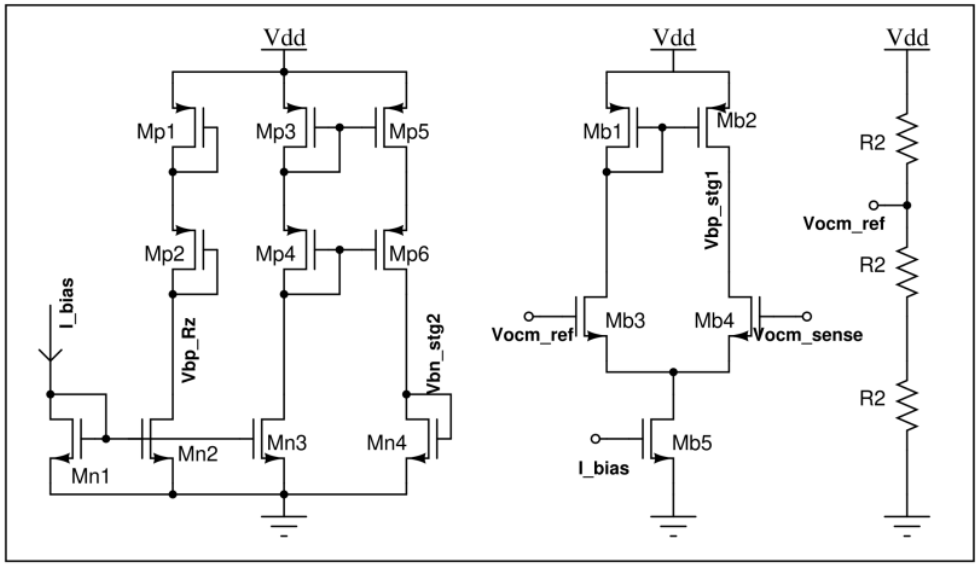

Figure 3.3: 2-Stage Differential Amplifier Netlist (b)

Then, we selected a 2-stage differential amplifier with tracking bias for the study. Figures 3.2 and 3.3 represent the netlist of a 2-stage Differential Amplifier with biasing circuits and a common-mode feedback circuit. We needed to define the input variables for this circuit, like the previous one, and the output specification remained the same, only the calculations needed for this circuit increased. Although it covers many applications, this circuit topology is slightly uncommon to be used by designers while designing and also contains complex calculations to be done before designing. Hence, this circuit was also not taken further. But the netlist and all relevant testbenches were developed and are used in further stages. One more reason for discarding this circuit was that it was too specific. We needed to introduce many variables to generalize the circuit's application, which might create confusion in creating the dataset in further stages. We might land on the same problem we faced in project phase 1. Hence, we selected a simpler circuit version, i.e., a single-ended 2-stage Operational Amplifier with tracking bias.

3.3 Final Circuit

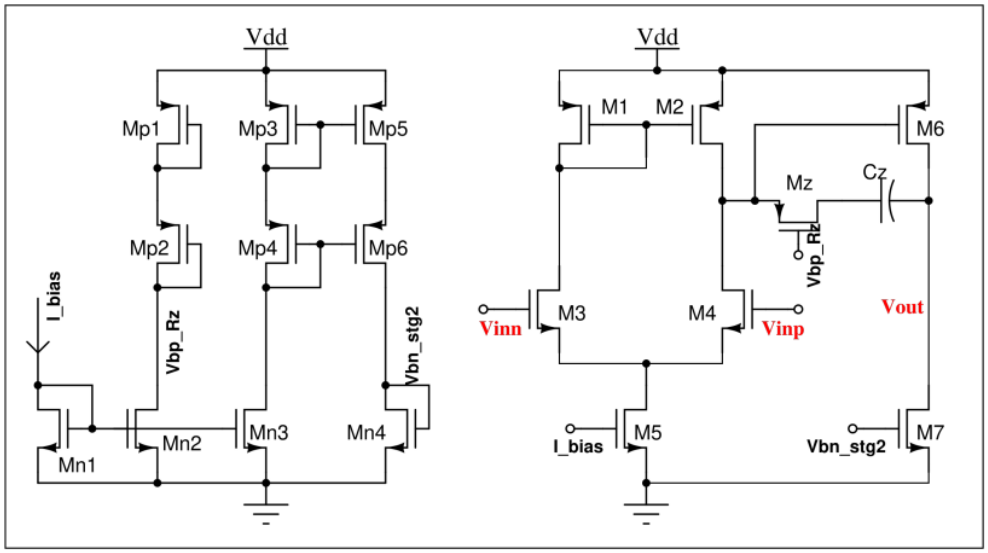

Figure 3.4: 2-Stage Single-Ended Operational Amplifier with Tracking Bias Netlist

The modified version of the previous circuit was still not final for proper analysis. We

removed the common-mode feedback circuit from the netlist, kept all the bias-generating circuit parts, and modified the circuit to a single-ended version with tracking bias. Figure 3.4 shows the netlist of a single-ended operational amplifier with tracking bias.

The next step was to perform hand calculations, create a netlist and testbenches, and do simulations to verify whether the design was working correctly. The relationships between circuit parameters and output specifications are discussed briefly in section 3.4.

Figure 3.5 represents the final netlist currently being used. After performing calculations, we did this modification from Figure 3.4. It includes fewer input variables with the same functionality. The main objective was not to create a single netlist with exact specifications as desired but to create multiple netlists with reached specifications that are near as required by the user.

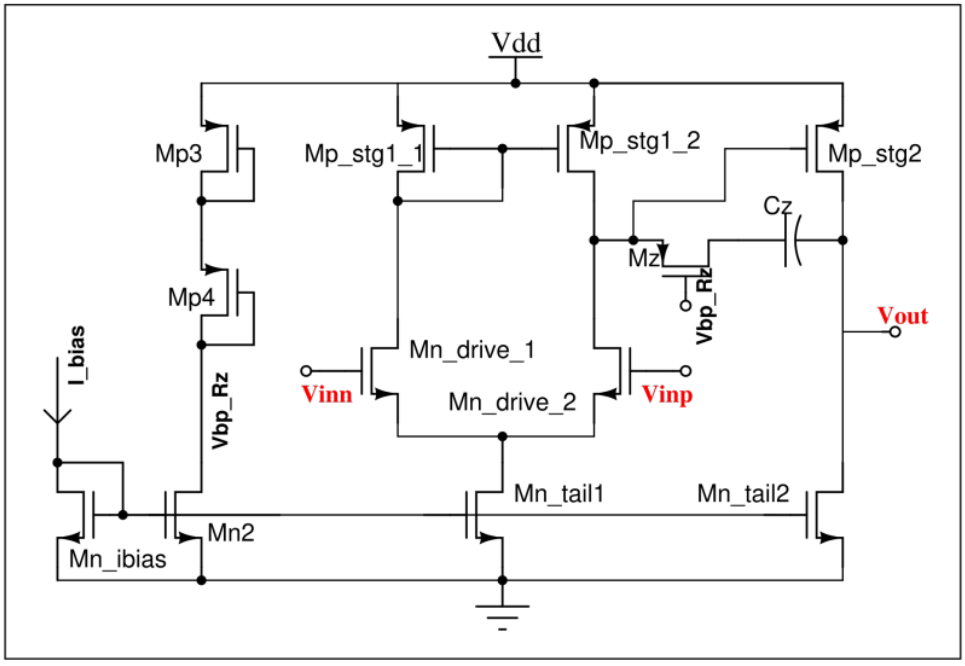

Figure 3.5: Modified 2-Stage Single-Ended Operational Amplifier with Tracking Bias Netlist

3.4 Literature Survey

After finalizing the design, the main task was to study the relationships between the inputs and outputs of the design. Also, we were required to finalize the input and output variables for the circuit. Hence, we did a literature survey for circuits represented in Figure 3.1 and Figure 3.5. Since the former is the basic circuit topology, we must understand its functionality to proceed further. And the latter one represents the final chosen/selected circuit. [5] summarizes the basic design of a two-stage single-ended operational amplifier. It provides some relations between inputs and outputs, few constraints in circuit parameters, etc. This study helped a lot in understanding the basic functionality of the circuit represented in Figure 3.1.

3.4.1 Previous Designs

Many state-of-the-art techniques over the past few decades show that the most common circuit topology for a 2-stage single-ended operational amplifier is similar to the circuit represented by Figure 3.1 as given in [5]. The only difference the netlists have is that some are based on n-MOS transistors for taking the differential input, while the rest are based on p-MOS transistors for taking the differential input.

The circuit represented by Figure 3.5 is rare to be used by designers since it includes tracking bias topology. Few researchers have given a brief study about this circuit. This circuit is used because it does not change its operation much on temperature variation. The temperature variation occurs due to the resistor present in the Miller Compensation feedback path. Hence, it is required to get rid of the resistor. The only method to proceed with this design is to place a transistor instead of a resistor and bias the transistor in the linear region. [3] briefly provides the summary of the design topology with p-MOS transistors being used for taking inputs of the differential amplifier.

3.4.2 Tracking Bias Method

Tracking Bias or Tracking RC Frequency Compensation technique is specifically used to avoid the transition of the region of operation of the transistor from linear to any other region. This method helps keep the circuit operating as desired, even on temperature variations. [3] and [4] specifically explain the working of this methodology. The transistor in the linear region operates as a resistor while keeping the sizes of that transistor as a function of other circuit parameters independent of temperature variation. Figure 3.6 shows the part of the circuit used to implement the tracking bias method.

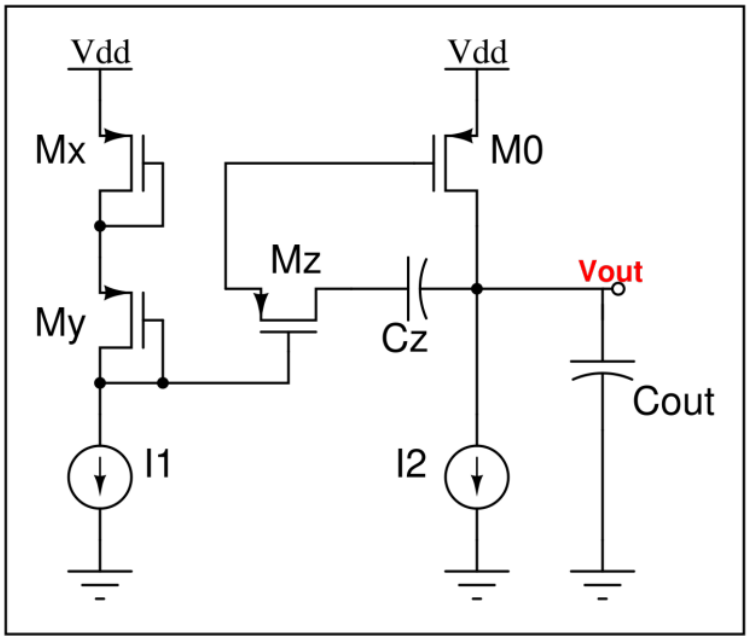

Figure 3.6: Tracking RC Frequency Compensation Netlist

$$\left(\frac{W}{L}\right)_z = \sqrt{\frac{I_2}{I_1}} \times \sqrt{\left(\frac{W}{L}\right)_0 \times \left(\frac{W}{L}\right)_y} \times \left(\frac{C_z}{C_z + C_{out}}\right) \tag{3.1}$$

Equation 3.1 gives the relationship of sizes for the transistor operating in the linear region.

We can see in the equation 3.1 that none of the variables are dependent upon temperature variation. Hence, we proved the motive of choosing the tracking RC bias methodology. By this, we complete the part of proper circuit selection and netlist definition. Before switching to machine learning, the final part of designing the circuit schematic and testbenches in Cadence Virtuoso remains.

3.5 Circuit Designing

After finalizing the netlist, we made the circuit represented in Figure 3.4, and Figure 3.7 represents the instance of the same circuit developed in the 65 nm technology process and supply voltage equal to 1.2 V.

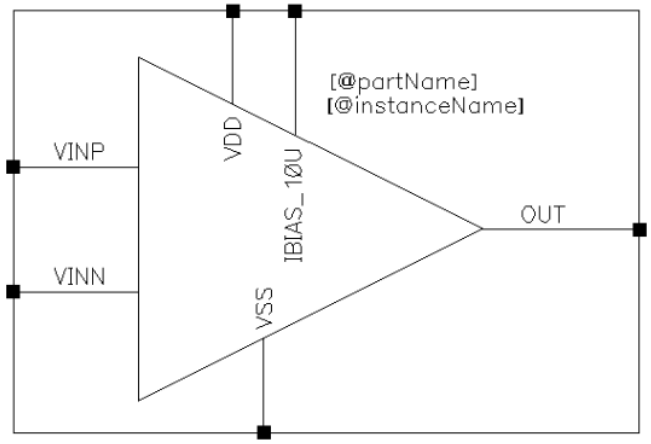

Figure 3.7: 2-stage Single-ended Operational Amplifier Instance

The testbenches designed for this circuit are as follows:

1. Stability Analysis (Figure 3.8
2. Slew Rate Analysis (Figure 3.9
3. Noise Analysis (Figure 3.10
4. CMRR Analysis (Figure 3.12

5. PSRR Analysis (Figure 3.11

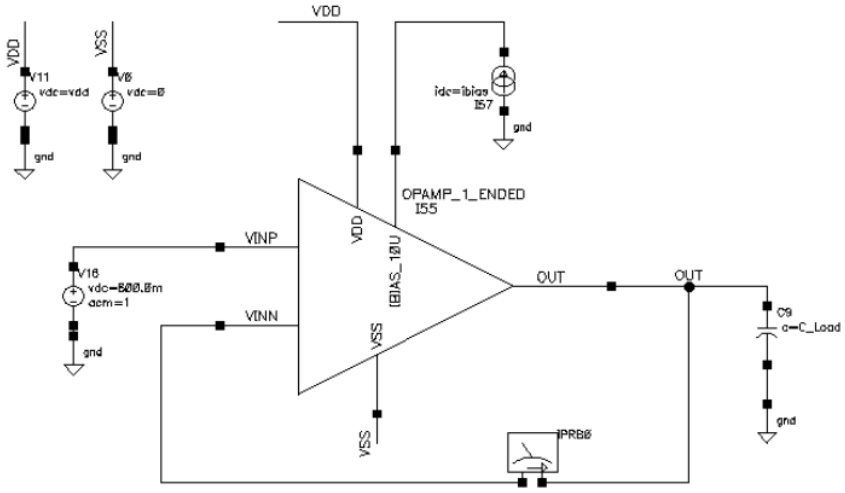

Figure 3.8: Testbench - Stability Analysis

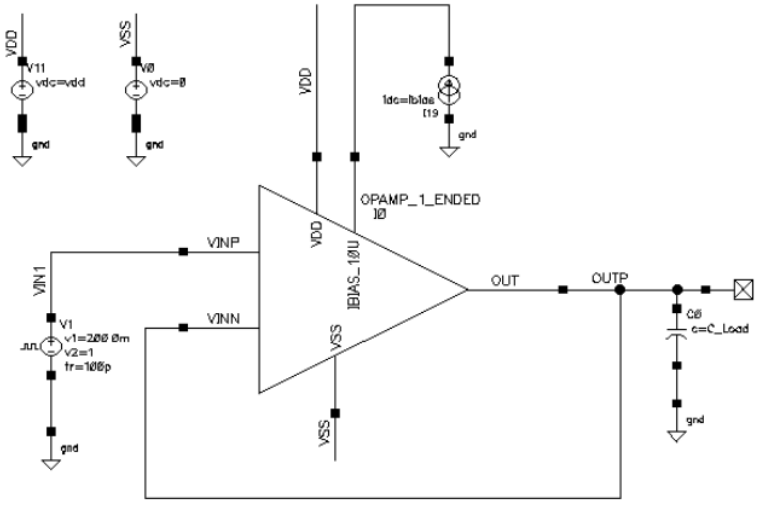

Figure 3.9: Testbench - Slew Rate Analysis

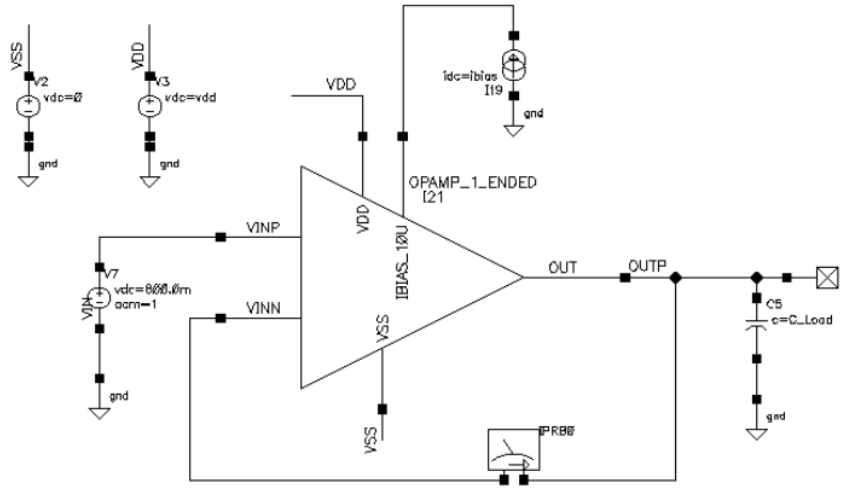

Figure 3.10: Testbench - Noise Analysis

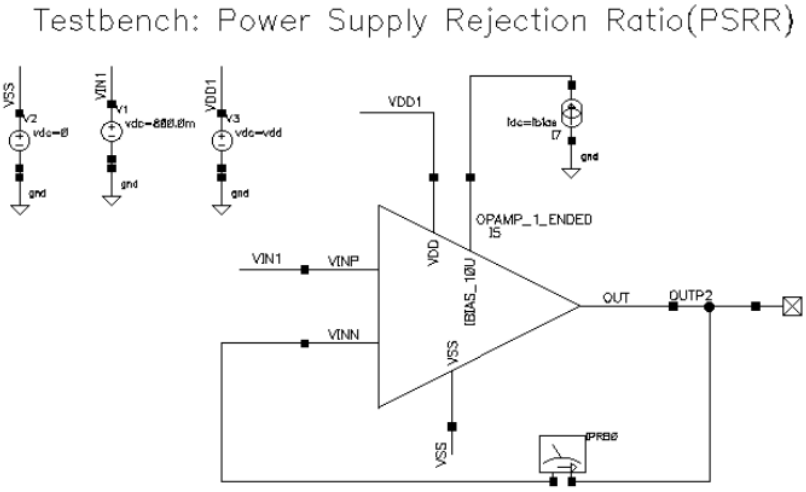

Figure 3.11: Testbench - PSRR Analysis

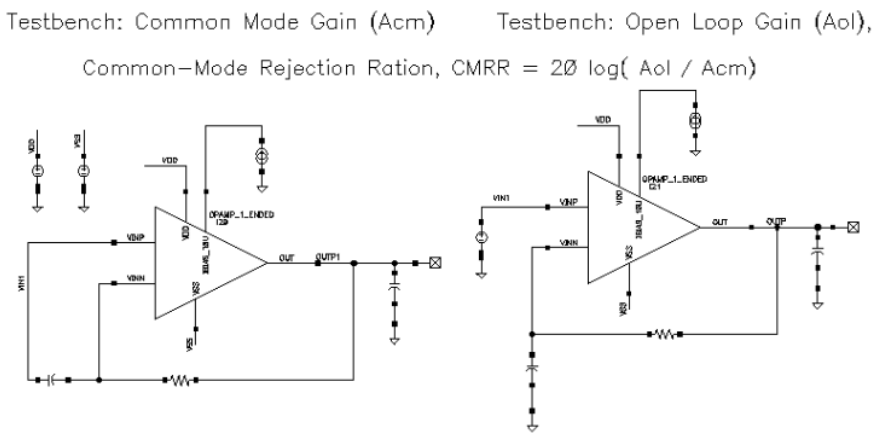

Figure 3.12: Testbench - CMRR Analysis

We obtain numeric outputs and waveform outputs from these testbenches. The requirement for this project is to extract numeric outputs from the design. Machine Learning Tool will use the numeric outputs for further project stages. In later stages, we will use these defined output variables. The Output variables are as follows:

1. DC Gain
2. DC Power
3. Phase Margin
4. Unity Gain Frequency
5. 3dB Bandwidth
6. Slew Rate
7. Noise at DC
8. Noise at 1 MHz
9. Noise at 10 MHz

We have now defined the output variables properly. We will deal with the relationship between these output variables later. Hence, we can now focus on defining the input variables.

Chapter 4

Challenges in Designing the Circuit

We now have a circuit netlist, circuit design, working testbenches, and well-defined output variables. But still, we don't have input variables that will vary for creating a dataset for the machine learning tool. We have a basic design with one set of independent values of widths, lengths, number of fingers, and multipliers for each transistor in the netlist. There is no relation between lengths of transistors acting as current mirrors, no relation between sizes of driving transistors, no relation between widths of any of the transistors, etc. Each acts as a separate entity, and we need to connect them via formulas, which is not acceptable.

4.1 Converting Sizes into Variables

We can understand the first iteration of converting sizes into variables as converting the sizes of highlighted transistors as shown in Figure 4.1 directly to variables.

Table 4.1: Variables in Design after Iteration 1

Lengths	**Widths**	**Multipliers**
L_Mn_drive	W_Mn_drive	mul_Mn_drive
L_Mn_tail1	W_Mn_tail1	mul_Mn_tail1
L_Mp_stg1	W_Mn_tail2	mul_Mn_tail2
	W_Mn_ibias	mul_Mn_ibias
	W_Mp_stg1	mul_Mp_stg1
	W_Mp_stg2	mul_Mp_stg2

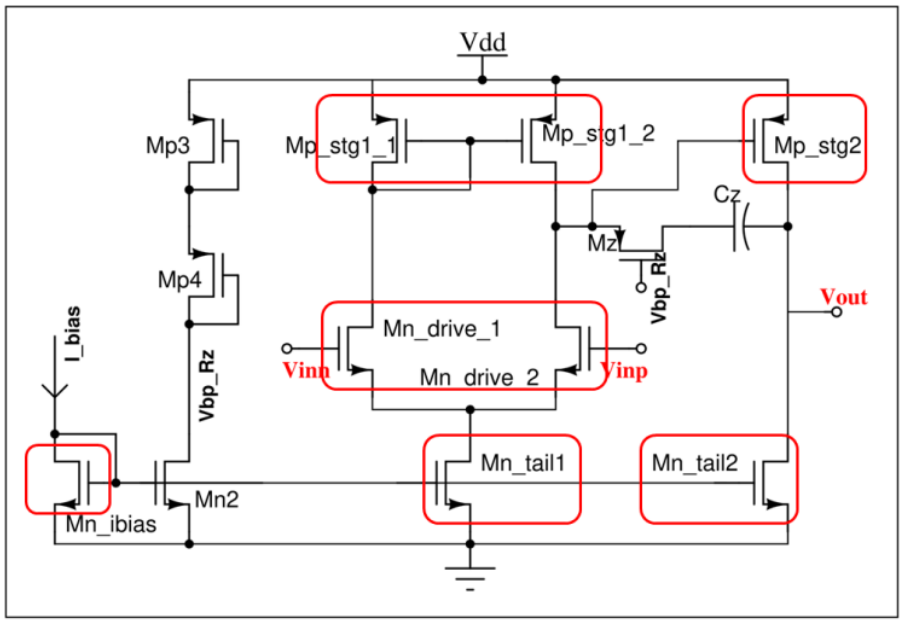

Figure 4.1: Converting Sizes to variables - Iteration 1

Table 4.1 shows the names of variables that we can select for the process. But since many unvaried/constant circuit parameters are also in the circuit, we would not be able to analyze the circuit outputs properly. Hence, we need to think as analog circuit designers and find the optimum set of variables. Therefore, we broke down variables for a single transistor into four variables, namely number of fingers, Finger Width of the transistor, Length of Transistor, and multiplier of the transistor. These variables are shown in Equation 4.1. Now, these variables can be dealt with for inter-relating the parameters of each transistor and selecting variables for the machine learning part. There also would be a requirement for minimizing the number of variables.

$$\textbf{total aspect ratio}, \frac{\mathbf{W}}{\mathbf{L}} = \left(\frac{\textbf{number of fingers} \times \textbf{finger width}}{\textbf{length of transistor}} \right) \times \textbf{multiplier of transistor} \tag{4.1}$$

Hence, for the second iteration of converting sizes into variables, equation 4.1 was considered. Since, at the time of designing, an analog designer might not specify the actual width

of the transistor, we can specify in terms of the number of fingers, finger width, and multipliers. Even if we fix the finger width for all transistors to keep the entire design with the same finger width for taking care of post schematic processes, we make the designer's effort a lot simpler. In this project, we followed the same method, and the finger width of all the transistors was fixed to 500 nm or 0.5 μm.

Table 4.2: Transistor Sizes in terms of Fixed Values or Variable Names

Name of Transistor	**Number of Fingers**	**Finger Width**	**Length of Transistor**	**Multiplier of Transistor**
Mn_ibias	**nf_ibias**	0.5 μm	**L_Mn_tail1**	2
Mn_tail1	**nf_Mn_tail1**	0.5 μm	L_Mn_tail1	2
Mn_tail2	**nf_Mn_tail2**	0.5 μm	L_Mn_tail1	2
Mn_drive_1	**nf_Mn_drive**	0.5 μm	**L_Mn_drive**	2
Mn_drive_2	nf_Mn_drive	0.5 μm	L_Mn_drive	2
Mp_stg1_1	**nf_Mp_stg1**	0.5 μm	**L_Mn_stg1**	2
Mp_stg1_2	nf_Mp_stg1	0.5 μm	L_Mp_stg1	2
Mp_stg2	**nf_Mp_stg2**	0.5 μm	**L_Mp_stg2**	4
Mp_Rz	**nf_Rz**	0.5 μm	L_Mn_stg2	2
Mp3	**nf_Mp3**	0.5 μm	L_Mn_stg2	2
Mp4	**nf_Mp4**	0.5 μm	L_Mn_stg2	2
Mn2	nf_ibias	0.5 μm	L_Mn_tail1	2

Table 4.3: Variables in Design after Iteration 2

Circuit Parameter	**Variable Name**
Number of Fingers	nf_ibias, nf_Mn_tail1, nf_Mn_tail2, nf_Mn_drive, nf_Mp_stg1, nf_Mp_stg2, nf_Mp_Rz, nf_Mp3, nf_Mp4
Lengths of Transistors	L_Mn_tail1, L_Mn_drive, L_Mp_stg1, L_Mp_stg2

The same can be the case with multipliers of transistors. If we fix the value of the multiplier for each transistor, whether the same or different, it will affect the number of variables by a huge factor. Table 4.2 gives the values of different variables set for each transistor. It can be either a fixed value representing a value set in design in Cadence Virtuoso software or a variable name used to represent the sizes of the particular transistor.

Apart from variables specified in tables 4.3, we have other variables unrelated to transistors. These variables are values of current supplied for generating bias voltage (i_{bias}), the value of coupling capacitor or compensation capacitor (C_Z), and the value of load capacitance (C_{load}). Hence, there can be **nine variables as the number of fingers of transistors, four variables as the length of transistors, and three non-transistors variables, a total of 16 variables**.

4.2 Minimizing Variables

We cannot directly use these 16 variables to generate a dataset as there are still inter-relations between them. Hence, we need to minimize the number of variables.

1. **Length Variables** -

   The Length variables need to be different for different parts of the circuit. The driving transistors or the transistors where the input of Op-Amp is given (Mn_drive_1 and Mn_drive_2) should have the same length variable. The transistors with current mirrors from biasing transistors (Mn_ibias, Mn_tail1, Mn_tail2, and Mn2) should have the same lengths. As for the different length variables for PMOS of the first and second stages of the op-amp, we can reduce a variable by keeping the same variable name for them. Hence, now we have three length variables at the final stage, which are **L_Mn_drive, L_Mn_tail1, and L_Mp_stg1**.

2. **Variables for Number of Fingers of Transistors** -

   #Fingers represents the Number of Fingers

   The most redundant variables are #Fingers. For transistors Mn_ibias, Mn_tail1, Mn_tail2, Mn_drive_1, and Mn_drive_2, the variable is necessary and can't be reduced as they exhibit separate functionality. But, we can interlink the rest variables in #Fingers with

one another and merge them into a single variable. The single variable chosen here is nf_Mp_st2, and the section 4.3 gives the relations of all other variables of #Fingers. Hence, the #Finger variables at final stage are **nf_ibias, nf_Mn_drive, nf_Mn_tail1, nf_Mn_tail2, and nf_Mp_stg2**.

3. **Other Variables** -

   Since we are making the design for Op-amp and not for load dependency, we can fix the capacitance variable (C_{load}) to a particular value. The input current variable (i_{bias}) is independent of other variables, and we can fix it to a single value at this stage and vary it in later stages. Hence, we only have one variable: coupling capacitance value (C_c). We can vary it by setting the multiplier of the capacitor as a variable. Hence, the other variables at the final stage are **i_bias and C_c**.

$$Coupling\ Capacitance,\ C_c = multiplier\ of\ capacitor\ \times\ minimum\ capacitance. \quad (4.2)$$

Table 4.4: Final Input and Output Variables

Input Variables	**Output variables**
nf_ibias	DC Gain
nf_Mn_drive	DC Power
nf_Mn_tail1	Slew Rate Value
nf_Mn_tail2	Phase Margin
nf_Mpn_stg2	Unity gain Frequency
L_Mp_stg1	3dB Bandwidth
L_Mn_tail1	Noise at DC
L_Mn_drive	Noise at 1 MHz
mul_cap	Noise at 10 MHz
ibias	

Please refer to the variables' names from Figure 3.5. We finally have the input and output variables represented in Table 4.4, and we are ready to generate the dataset.

4.3 Relation between Parameters

This section gives the formulas used to generate the output variables in the Cadence Virtuoso environment. It also gives the relations between chosen input variables to minimize the number of input variables.

The formulas for Output Variables are given below.

$$DC\ Gain = value(db(mag(getData(\text{"}loopGain\text{"}\ ?result\ \text{"}stb\text{"}))))\ 0.1) \tag{4.3}$$

$$Phase\ Margin = getData(\text{"}phaseMargin\text{"}\ ?result\ \text{"}stb_margin\text{"}) \tag{4.4}$$

$$3dB\ Bandwidth = bandwidth(dB20(mag(getData(\text{"}loopGain\text{"}\ ?result\ \text{"}stb\text{"}))))\ 3\ \text{"}low\text{"}) \tag{4.5}$$

$$Unity\ Gain\ Bandwidth = unityGainFreq(mag(getData(\text{"}loopGain\text{"}\ ?result\ \text{"}stb\text{"})))) \tag{4.6}$$

$$DC\ Power = (-1 * OP(\text{"}/V11\text{"}\ \text{"}pwr\text{"})) \tag{4.7}$$

$$Slew\ Rate = slewRate(VT(\text{"}/OUTP\text{"})\ 0.4\ nil\ 1\ nil\ 5\ 95\ nil\ \text{"}time\text{"}) \tag{4.8}$$

$$Noise\ at\ DC = value(getData(\text{"}out\text{"}\ ?result\ \text{"}noise\text{"})\ 0.1) \tag{4.9}$$

$$Noise\ at\ 1MHz = value(getData(\text{"}out\text{"}\ ?result\ \text{"}noise\text{"})\ 1000000) \tag{4.10}$$

$$Noise\ at\ 10MHz = value(getData(\text{"}out\text{"}\ ?result\ \text{"}noise\text{"})\ 10000000) \tag{4.11}$$

The relations between transistor sizes are given below.

$$nf\_ibias\ =\ int(1 * L\_Mn\_tail1/1u) \tag{4.12}$$

$$nf\_Mn\_drive\ =\ int(nf\_Mn\_drive\_values * L\_Mn\_drive/400n) \tag{4.13}$$

$$nf\_Mn\_tail1\ =\ int(nf\_Mn\_tail1\_values * L\_Mn\_tail1/1u) \tag{4.14}$$

$$nf\_Mn\_tail2\ =\ int(nf\_Mn\_tail2\_values * L\_Mn\_tail1/1u) \tag{4.15}$$

$$nf\_Mp\_stg2 \;=\; int(nf\_Mp\_stg2\_values * L\_Mp\_stg1/200n) \tag{4.16}$$

$$nf\_Mp\_stg1 \;=\; int\left(\frac{nf\_Mp\_stg2 \times nf\_Mn\_tail1}{2 \times nf\_Mn\_tail2} \times \frac{L\_Mp\_stg1}{200nm}\right) \tag{4.17}$$

$$nf\_Mp3 \;=\; nf\_Mp4 \times 2 \tag{4.18}$$

$$nf\_Mp4 \;=\; 1 + int\left(\frac{nf\_Mp\_stg2 \times nf\_ibias}{nf\_Mn\_tail2} \times \frac{L\_Mp\_stg1}{200nm}\right) \tag{4.19}$$

$$nf\_Rz = int\left(\sqrt{\frac{nf\_Mn\_tail2 * nf\_Mp\_stg2 \times nf\_Mp3}{nf\_ibias}} \times \frac{51fF \times mul\_cap}{C\_Load + 51fF \times mul\_cap} \times \frac{L\_Mp\_stg1}{200nm}\right) \tag{4.20}$$

Chapter 5

Generating Datasets

At this project stage, we have everything ready for creating the Datasets for the project. The input variables are adequately studied, and the output variables needed are defined correctly. The Cadence Virtuoso environment has the setup to run multiple scripts to obtain multiple test points, which will act as datasets for the machine learning tool.

5.1 Variable Definition and Variation

Once again, we define the input and output variables, clearly specifying the set value of variables, if any.

- **Input Variables** -

    1. nf_ibias - This variable denotes the number of fingers of transistors “Mn_ibias” and “Mn2”, the transistors where we give the input current for generating bias voltages. This variable is kept constant at “1” and is changed only when the variable L_Mn_tail1 is changed to keep the aspect ratio constant.
    2. nf_Mn_drive - This variable denotes the number of fingers of transistors “Mn_drive_1” and “Mn_drive_2”, the transistors where we give the input of the op-amp.
    3. nf_Mn_tail1 - This variable denotes the number of fingers of transistor “Mn_tail1”, the tail transistor in stage 1 of the op-amp.
    4. nf_Mn_tail2 - This variable denotes the number of fingers of transistor “Mn_tail2”, the tail transistor in stage 2 of the op-amp.

5. nf_Mp_stg2 - This variable denotes the number of fingers of transistor “Mp_stg2”.
6. L_Mn_drive - This variable denotes the length of transistors “Mn_drive_1” and “Mn_drive_2”, the transistors where we give the input of the op-amp.
7. L_Mn_tail1 - This variable denotes the length of transistor “Mn_ibias”, “Mn2”, “Mn_tail1”, and “Mn_tail2”. Since these transistors are in the current mirror configuration, we keep the length variable value same.
8. L_Mp_stg2 - This variable denotes the length of all PMOS transistors in design. We keep the length of all PMOS transistors the same because they are interrelated.
9. mul_cap - This variable denotes the multiplier of coupling capacitor “C_z” multiplier. We can vary the total value of C_z by varying this variable. The minimum value of C_z with a multiplier value equal to 1 is 51 fF.
10. ibias - This variable denotes the input current injected for generating bias voltages, although this variable is kept constant in this project, equal to 10 μA.
11. C_Load - This variable denotes the value of the load capacitor used for testing the design, kept constant at a value of 0.5 pF.
12. Supply Voltage, Vdd - The supply voltage used in this design is 1.2 V.

- **Output Variables** -

1. DC Gain - This output variable defines the op-amp gain at DC in the Stability Analysis testbench in Figure 3.8.
2. DC Power - This output variable defines the entire power usage by op-amp at DC in the Stability Analysis testbench in Figure 3.8.
3. Slew Rate Value - This output variable defines the slew rate value of op-amp in Slew Rate Analysis testbench in Figure 3.9.
4. Phase Margin - This output variable defines the Phase Margin of the op-amp in the Stability Analysis testbench in Figure 3.8.
5. 3dB Bandwidth - This output variable defines the 3db Bandwidth of op-amp in the Stability Analysis testbench in Figure 3.8.
6. Unity Gain Bandwidth- This output variable defines the Unity Gain Bandwidth of the op-amp in the Stability Analysis testbench in Figure 3.8.

7. Noise at DC - This output variable defines the Noise in op-amp design at DC in the Noise Analysis testbench in Figure 3.10.

8. Noise at 1 MHz - This output variable defines the Noise in op-amp design at 1 MHz in the Noise Analysis testbench in Figure 3.10.

9. Noise at 10 MHz - This output variable defines the Noise in op-amp design at 10 MHz in the Noise Analysis testbench in Figure 3.10.

5.2 Multiple Datasets Obtained

At this stage, we solved the problem of variables. But the range in which we can vary these is not defined because since an op-amp has to work with all transistors in saturation except for the transistor working in the linear region, we needed to ensure that every transistor's region of operation is saturation. Hence, we attempted to generate a dataset from the start point where the design is working.

5.2.1 Dataset 1

Table 5.1 shows the variables varied in this design. For each variable, we took three values and simulated all the possible combinations in Cadence Virtuoso. This approach gave a dataset with $3^5 = 243$ combinations. Out of these 243 combinations, 126 combinations passed the criteria of all transistors in saturation. Hence, we used it to train the machine learning model and proceed with further stages. But this dataset does not include variation in all variables; hence it has limitations.

Table 5.1: Dataset 1

Variable Name	nf_ibias	nf_Mn_drive	nf_Mn_tail1	nf_Mn_tail2	nf_Mp_stg2
Values	2, 4, 8	13, 26, 53	22, 44, 88	115, 230, 460	216, 432, 864

5.2.2 Dataset 2,3, and 4

Then, to add more variables to the dataset, variables “ibias” and “mul_cap” were also included. Although this step gave a large dataset, it took a long time to get simulated on cadence virtuoso. Therefore, we did a trial with only two values of all the variables, giving 512 combinations. As failed, we obtained the dataset with more than $(2/3)^{rd}$ of it.

Then an attempt to generate a dataset with all the length variables to be left out was made. Table 5.2 represent the variables and their values chosen this time.

Table 5.2: Dataset 4

Variable Name	**Values**
nf_ibias	2, 4, 8
nf_Mn_drive	13, 26, 53
nf_Mn_tail1	22, 44, 88
nf_Mn_tail2	115, 230, 460
nf_Mp_stg2	216, 432, 864
mul_cap	6 : 2 : 14
ibias	5 μA : 3 μA : 20 μA

5.3 Final Dataset Explanation

Finally, we developed a strategy to generate an appropriate dataset, which includes all the variables. It was such that we could vary the ’Length variables’ at a time, keeping all other variables constant and verifying the working with multiple sets of constant values of left-out variables. We are then varying all the rest variables, keeping all the length variables constant in value, and finally merging the two to obtain one sizeable adequate dataset. Also, we fixed the variable "ibias" to a constant value. Table 5.3 gives the values of the Length variables selected, and Table 5.4 gives the multiple sets of constant values chosen for the rest of the variables.

Table 5.3: Values for Length Variables in Final Dataset

Variable Name	**Range/Values**
L_Mn_drive	200 nm : 25 nm : 500 nm
L_Mn_tail1	1 μm : 100 nm : 3 μm
L_Mp_stg1	200 nm : 50 nm : 450 nm

Table 5.4: Multiple Sets of Constant values for Rest Variables

Datasets	**Dataset 5**	**Dataset 6**	**Dataset 7**
nf_ibias	1	1	1
nf_Mn_tail1	10	8	12
nf_Mn_drive	26	22	32
nf_Mn_tail2	60	50	72
nf_Mp_stg2	216	180	260
mul_cap	10	10	10
ibias	10 μA	10 μA	10 μA

Table 5.5: Dataset 8

Variable Names	**Values/Range**
nf_Mn_tail1	2 : 6 : 32
nf_Mn_drive	2 : 7 : 72
nf_Mn_tail2	20 : 40 : 260
nf_Mp_stg2	100 : 100 : 1000
mul_cap	6 : 2 : 14

This approach provided us with three datasets, each with 1638 combinations where only the length of each transistor is variable, and the other variables in the design are constant. Table 5.5 gave the values of other variables when length variables were kept constant.

L_Mn_drive = 400 nm, L_Mn_tail1 = 1 μm, L_Mp_stg1 = 200 nm, ibias = 10 μm, nf_ibias = 1

Dataset 8 contained 23100 combinations. After merging Datasets 5,6,7 and 8, we had 28014 combinations. It contains all the variables that were varied appropriately and have both pass and fail combinations. Table 5.6 gives the final details of the dataset used for the machine learning model used in this dataset. Hence, we have a total of **13977** valid combinations of simulations on which we can apply machine learning algorithms.

Table 5.6: Valid Combinations in Final Dataset

	Pass	**Fail**	**Total**
Dataset 5	1610	28	1638
Dataset 6	1360	268	1638
Dataset 7	1541	97	1638
Dataset 8	9466	13634	23100
Total	**13977**	14037	28014

The criteria for pass/fail of a simulation was set to:

- All transistors should be in Saturation region, except "Mn_Rz" from Figure 3.5.
- Phase Margin of Design should be greater than 45°.

Chapter 6

Machine Learning Model

I created the project's machine learning model using **Python** in **Spyder Environment**. The need for python as a programming language in this project is that it is very user understandable and easy to work with. Working with variables and machine learning can be done efficiently without taking much care of memory consumption, speed of operation, and syntactical mistakes.

6.1 Libraries Used

The libraries used in this project are enumerated below.

1. **time** - To calculate the time for processing different part of the code
2. **numpy** - To do mathematical operations on arrays and vectors
3. **pandas** - To handle python dictionaries and operation on dataset
4. **sns, matplotlib** - For visualization of plots and graphs
5. **sklearn** - For creating machine learning models, data processing, hyper-parameter tuning and cross-validation and computing results
6. **PyQt5** - For creating UI window
7. **sys, os** - For handling systems related arguments
8. **pickle** - For saving/loading models which take time to get created

6.2 Data Cleaning, Data Scaling and Pre-processing

After finally obtaining the dataset in section 5.3, data processing is a necessary process. Since we obtained the dataset from another software, i.e., Cadence Virtuoso, it is not ready to be fed to the machine learning model as it is. Thus, we need to do the data cleaning and processing.

6.2.1 Data Cleaning

The comma-separated value (.csv) file obtained from cadence contains extra columns that are not required or are unnecessary. We need to remove them first. Then, the simulation results for failed combinations need to be removed, after which we obtain the dataset ready to be processed by the machine learning tool. Now the values of variables are to be considered since some variables like 'length of transistors' and 'Noise' have values within the range 10^{-7}, while the number of fingers ranges from 100 to 1000. hence, the variable values need to be scaled appropriately.

6.2.2 In-built Scaling Functions Not Usable

Many scaling functions are available in python library **sklearn**, but we cannot directly use those scalars here. Direct Scaling functions lose the meaning of variable value and often give inaccurate results. The name of scaling functions and the reasons why we can't use them here are given below.

1. Standard Scalar -

   This scaling function scales the values of variables to a definite range of [-1,1], where the mean of values is 0. But it decreases the significance of input variables in this project. If we give an input variable a value that is equal to its mean in the dataset, then it will lose its significance. Hence, we can't use Standard Scalar.

2. Minmax Scalar -

This scaling function scales the values of variables to a definite range from minimum of all variables to maximum of all variables, which is quite significant in this project, due to which we have to use scaling functions. It does not solve the problem of scaling the variable values but further increases it.

Hence, we must manually scale all the variables according to the specific ranges. We need to study all the variables in terms of their ranges and develop their manual scaling function.

6.3 Support Vector Regression Technique

A Support Vector Machine model represents the instances as points in space, mapped so that the examples of the different categories are separated by as wide a gap as feasible. New instances are then mapped into the same space and classified according to which side of the gap they land. In a high- or infinite-dimensional space, a support vector machine creates a hyperplane or group of hyperplanes for classification, regression, and other tasks. The hyperplane with the most significant distance to the nearest training data points of any class (so-called operating margin) achieves a decent separation because the higher the margin, the lower the classifier's generalization error.

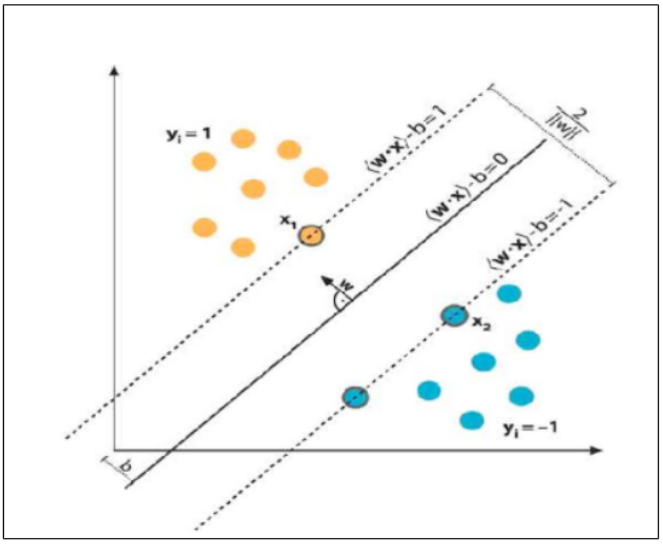

Figure 6.1: Support Vector Machine Hyperplane

In this project, the Support Vector Machine is used for regression since we have to predict the values of variables based on specific input values. The regression-based technique takes a vector of input variables for training. It puts the output variable value on an n-dimensional space, where n is the number of input features. Since we use regression for predicting, it works on multi-class classification, with each class having minimal boundary to predict the output variable accurately. The regression technique leans with every input given to it and works on decreasing the error by applying a penalty to misclassified points. Users can set the tolerance of points classified out of boundary. The user must set many other variables before starting the training process. These parameters are called Hyperparameters, and we need to tune them to get accurate classification results on unseen data. The output obtained after the training process is attribute weights, which signify the importance of each input in predicting the output classes.

6.4 Default Values of Hyper-Parameters

The hyperparameters are the set of parameters used to set the model's accuracy. These parameters already have a default value and can be directly used to train the model without specifying the parameters in the code. But if we are not satisfied with the model's accuracy, then we may change the values from default ones to custom values and check if there is any improvisation in accuracy. Table 6.1 gives the default values of parameters used in the code.

Table 6.1: Default Values for various Hyper-Parameters

Hyperparameter Name	**Default Values**
kernel	rbf (radial basis function)
Kernel Co-efficient, **gamma**	1/(number of features)
tolerance for stopping criterion	0.001
Regularization Parameter, **C**	1
epsilon	0.1

6.5 Selection of Hyper-Parameters for All Variables

During the model training stage, the parameter's default values were not giving accurate results. Thus, there was a necessity for Hyperparameter tuning. Also, we cannot try any random value for the set of hyperparameters since our dataset is large enough, and it takes weeks for trying all the sets of values to be fed manually. And even after trying different sets of values and testing on the unseen data, we can't measure the accuracy of the created model. Therefore, by Hyperparameter tuning and Cross-Validation technique, the problems of unseen data testing and accuracy check can be checked and solved. Figure 6.2 gives a block diagram for hyperparameter tuning and cross-validation. Table 6.2 gives the values of each Hyperparameter chosen for tuning.

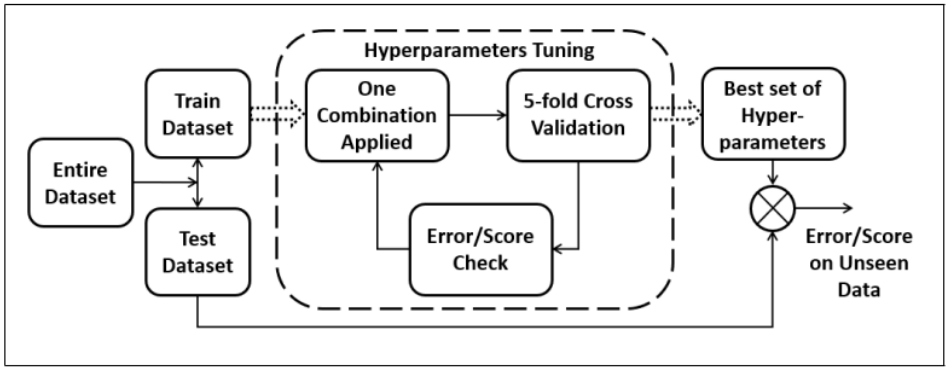

Figure 6.2: Hyperparameter Tuning and Cross-Validation

Table 6.2: Set of Values for Hyper-Parameter Tuning

Hyperparameter Name	Set of Values selected
kernel	linear, rbf, poly, sigmoid
gamma	1, 0.1, 0.01, 0.001
tol	0.1, 0.01, 0.001
C	0.1, 1, 10, 50
epsilon	1, 0.1, 0.01, 0.001

Chapter 7

Testing of Machine Learning Model

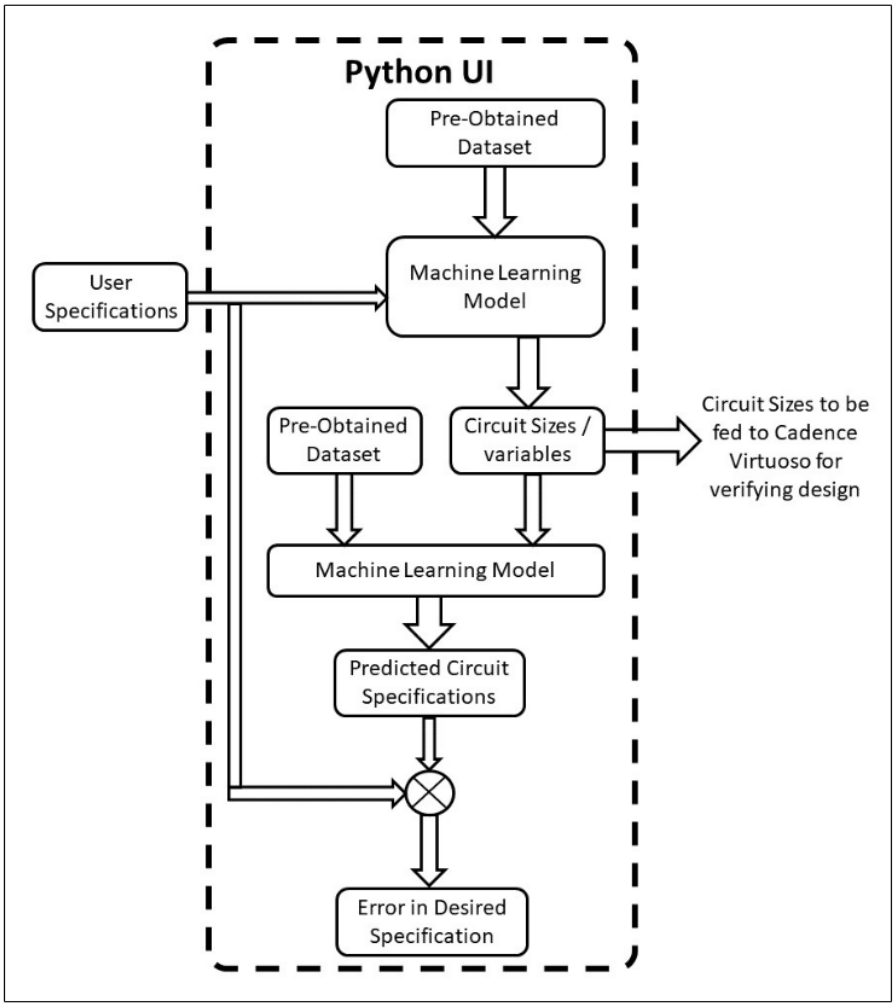

Figure 7.1: Working of designed ML Tool

We have created the machine learning tool. Figure A.1 briefly describes the working of the tool. In this, the specification of the circuit from the user is the primary input. We take the specifications as an input vector, on which the model created using the training dataset works to predict the Circuit Sizes/Variables. The circuit sizes are the output of the tool. These output values are to be taken by the user and fed into the circuit design on Cadence Virtuoso software to ensure that we obtain the desired specifications.

The tool does specifications based on the original training dataset and another machine learning model. The final predicted result is then compared with the desired circuit specification, finally providing the error. The main objective of this project is to minimize this error. If the error in predicting the desired specifications becomes negligible, then providing the circuit sizes to the user reduces the design time of the circuit for any given specifications.

7.1 Errors during Hyper-Parameter Tuning

Following the created ML tool, there is some error at every point where we take machine learning output, and the error can get multiplied if we take erroneous output further. Hence, errors at every stage must be studied and minimized to the possible extent. During the creation of models, selecting hyperparameters is a crucial step, as other than the best parameters may lead to inaccurate results. Table 7.1 gives the values of the best set of hyperparameters for each input and output variable chosen in this design.

The entire process of Hyperparameter tuning takes at least a day or two. Once it gets finished, we give the best parameters obtained to the model and take the score on the test dataset. The time duration to complete hyperparameter tuning depends on the number of data points in the dataset and the variation in data points. This project's dataset took around 34 hours for one round of tuning for all the variables. If some error occurs, or there is some code calculation error, the output results will be wrong, and we need to perform the whole tuning process again.

Figure 7.2 and Figure 7.3 give the values of scores obtained for each input and output variable after the model takes the best parameters for training. The score value depicted in this table is the measure of the accuracy of the created model. A score value equal to 1 denotes perfect predictions, while a score value equal to 0 denotes random prediction. There is a gradual

increase in score value starting from around 0.01 o 0.2 for each variable when we give the model the different sets of values for hyperparameters. The table below shows the final value reached after iterating the entire dataset.

Table 7.1: Best set of Hyperparameters for each Input and Output Variable

Variable Name	**Kernel**	**C**	**epsilon**	**gamma**	**tol**
Output Variables					
DC Gain	rbf	50	0.01	0.001	0.001
Slew rate Value	rbf	50	0.01	0.01	0.001
Unity Gain Bandwidth	rbf	50	0.01	0.01	0.001
Phase Margn	rbf	50	0.01	0.01	0.001
DC Power	rbf	50	0.01	0.01	0.001
3dB Bandwidth	rbf	50	0.01	0.01	0.001
Noise at DC	rbf	50	0.01	0.001	0.001
Noise at 1 MHz	rbf	50	0.01	0.01	0.001
Noise at 10 MHz	rbf	50	0.01	0.01	0.001
Input Variables					
nf_Mn_drive	rbf	50	0.01	0.01	0.001
nf_Mn_tail1	rbf	50	0.01	0.01	0.001
nf_Mn_tail2	rbf	50	0.01	0.01	0.001
nf_Mp_stg2	rbf	50	0.01	0.01	0.001
nf_ibias	rbf	50	0.01	0.1	0.001
L_Mn_drive	rbf	50	0.01	0.01	0.001
L_Mn_tail1	rbf	50	0.01	0.1	0.001
L_Mp_stg1	rbf	50	0.01	0.01	0.001
mul_cap	rbf	50	0.01	0.01	0.001

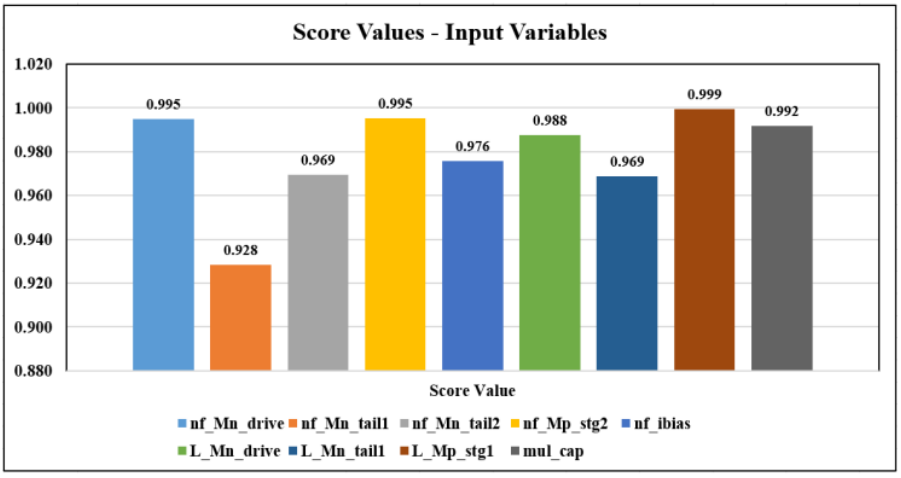

Figure 7.2: Score Values for Different Input Variables after Hyperparameter Tuning

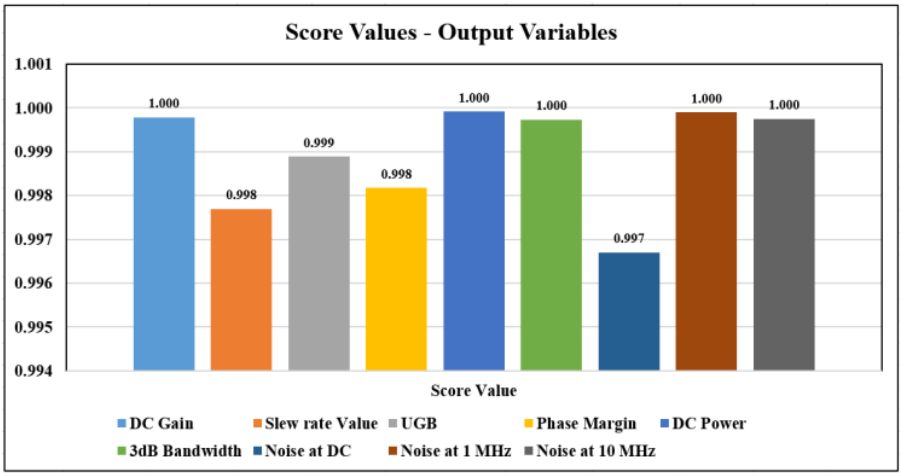

Figure 7.3: Score Values for Different Output Variables after Hyperparameter Tuning

7.2 Testing Model from a User Perspective

Now the model and tool are ready to be tested with different types of inputs. The input is given by a user or can be given from a user perspective. This method tells how the user wants the output to be received. Hence, two methods are discussed as follows.

7.2.1 Taking Input from Dataset

A user/designer may not know what specifications they should give the tool for testing. Hence, we take the input from the dataset itself as Train and Test Dataset. We create the models using the training dataset and take unseen data points from the testing dataset. The models with the best parameters obtained after cross-validation ensure that model can take test data from any part of the dataset. Hence, the accuracy is not compromised.

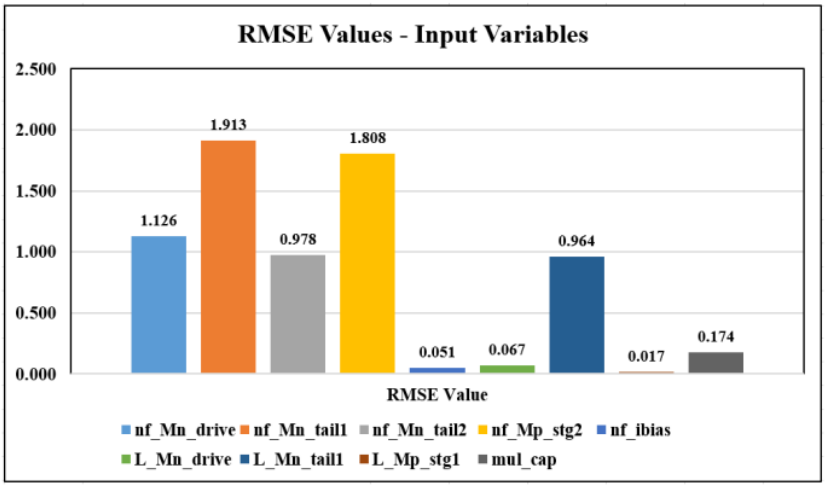

Figure 7.4: RMSE Values for Different Input Variables

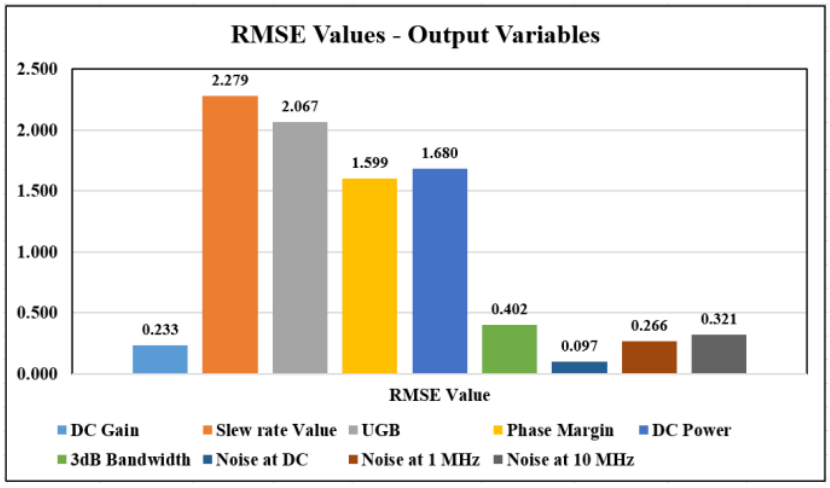

Figure 7.5: RMSE Values for Different Output Variables

Since we have the actual value of both circuit sizes and circuit specifications, the computation of error is an effortless task. The error is then calculated as Root Mean Square Error (RMSE) values for both inputs and outputs as represented in Figure A.1. Although the error for output variables is much more significant for us, we consider both. Figures 7.4 and 7.5 represents the results obtained in this method.

7.2.2 Taking Input from User

The other method of taking input from the user perspective is that we take the input from a real user. Figure 7.6 represents the user interface (UI) generated using Python. The user can give the values of the specifications as input. The press of **"Calculate"** button displays the result for circuit sizes. Pre-generated cadence netlist takes these circuit parameters for verification. **"Set Default"** button press sets all the specifications with some default value. Also, the **"Clear"** button press clears all the fields.

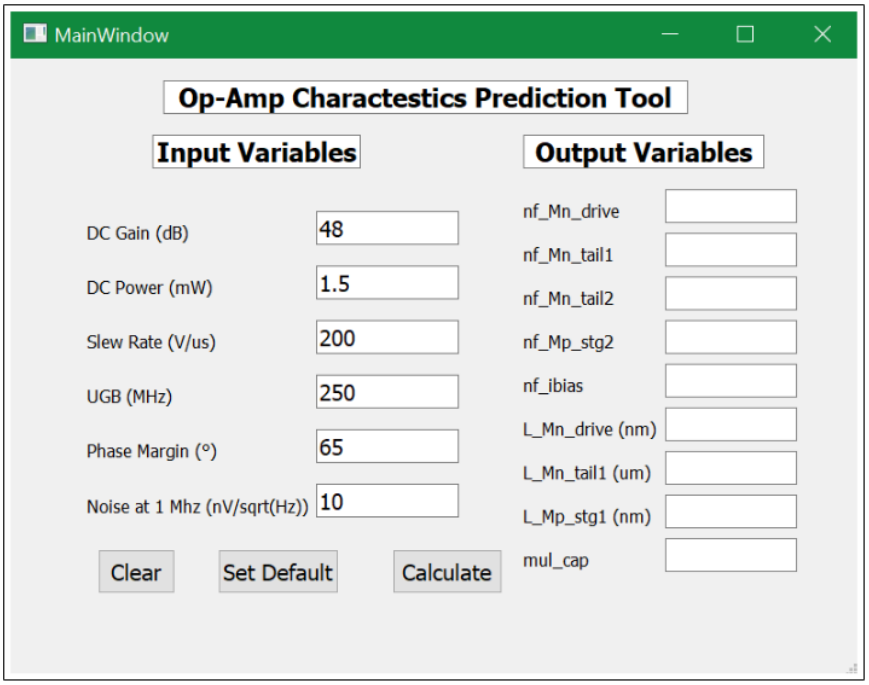

Figure 7.6: UI Window of created ML Tool

Chapter 8

Results, Summary and Future Work

Table 8.1, 8.2, and 8.3, gives the values of desired, actual and predicted specifications after feeding the circuit sizes into Cadence Virtuoso software.

Desired Value column denoted the values required by the user given in the UI window. The **Actual Value** column denotes the value of specifications obtained after feeding circuit sizes, that was, the output of the UI window, into the Cadence Virtuoso software. The **Value column** denoted the value of specifications obtained from the ML tool at the backend (checked as a variable in Python Software).

Table 8.1: Desired, Actual and Predicted Specifications - Set 1

Specifications	**Desired Value**	**Actual Value (Cadence Output)**	**Predicted Value (ML Tool Output)**
DC Gain (dB)	48	50.63	50.19
DC Power (mW)	1.5	0.859	0.868
Slew rate Value (V/μs)	200	141.2	133.66
Unity Gain Bandwidth (MHz)	250	162	160.93
Phase Margin ($^\circ$)	65	62.03	62.2
3dB Bandwidth (MHz)	4.55	2.7	3.03
Noise at DC (μV/$\sqrt{Hz}$)	33.32	24.13	29.83
Noise at 1 MHz (nV/$\sqrt{Hz}$)	10	9.232	10.23
Noise at 10 MHz (nV/$\sqrt{Hz}$)	7.261	8.19	8.79

Table 8.2: Desired, Actual and Predicted Specifications - Set 2

Specifications	**Desired Value**	**Actual Value (Cadence Output)**	**Predicted Value (ML Tool Output)**
DC Gain (dB)	45	48.53	48.02
DC Power (mW)	1	0.503	0.432
Slew rate Value (V/μs)	300	214	228
Unity Gain Bandwidth (MHz)	270	167.9	155.11
Phase Margin ($^\circ$)	60	52.7	53
3dB Bandwidth (MHz)	4.91	3.04	3.39
Noise at DC (μV/$\sqrt{Hz}$)	40	21.95	24.55
Noise at 1 MHz (nV/$\sqrt{Hz}$)	12	9.178	10.64
Noise at 10 MHz (nV/$\sqrt{Hz}$)	8.72	7.568	8.43

Table 8.3: Desired, Actual and Predicted Specifications - Set 3

Specifications	**Desired Value**	**Actual Value (Cadence Output)**	**Predicted Value (ML Tool Output)**
DC Gain (dB)	50	48.78	48.79
DC Power (mW)	1.5	1.624	1.575
Slew rate Value (V/μs)	400	271.9	250
Unity Gain Bandwidth (MHz)	500	314.2	318
Phase Margin ($^\circ$)	65	73.17	78.78
3dB Bandwidth (MHz)	9.10	5.37	5.58
Noise at DC (μV/$\sqrt{Hz}$)	33.32	45.67	45.57
Noise at 1 MHz (nV/$\sqrt{Hz}$)	10	10.11	10.4
Noise at 10 MHz (nV/$\sqrt{Hz}$)	7.261	7.49	7.78

Most of the specifications are getting predicted with significantly less absolute error. But we can see there are errors in accurately predicting the values for some specifications. The

following work is of an analog designer to get accurate values as desired by changing them according to their knowledge. As this tool is for Design Time Reduction and not removal, some effort from Analog Designer is still required to make the design perfect.

8.1 Conclusion

To conclude this project in one line, we have not created just a single circuit design of a 2-stage Single Ended Operational Amplifier. But collective knowledge of analog circuits and machine learning knowledge to create different circuit designs of the same circuit, with just a few input specifications from the user. This technique or method helps analog designers reduce the time and effort in designing the circuit. The work included here is required once, and the designers can experience the utility of this tool.

This process can be explained/summarized for any known analog circuit in following steps:

1. Circuit Selection
2. Netlist Definition
3. Designing the circuit for single specification
4. Conversion of Circuit Parameters to Variables
5. Defining Output Variables
6. Study of relationship between input and output variables
7. Generating Dataset for Input-Output Relationships
8. Creating Machine Learning model and its training process
9. Error Minimization for accurate results
10. UI development/modification
11. Testing the created tool

Following the above steps, we can design any analog circuit using machine learning techniques. Only the designing and dataset generation parts may take time during the process since the complexity of each circuit is different. It doesn't take long for simpler circuits to convert their circuit parameters into variables. Still, the more complex the circuit is, the more the circuit parameters and output variables will be. Hence, to conduct the design process of this type of tool, both analog circuit knowledge and machine learning knowledge are required.

We can verify the design of the circuit with parameters from the ML model up to the schematic level only. Also, we cannot make the model with very high accuracy since there can be variations in outputs when we take the design for further processes. Analog Designer's knowledge is required to perfect the design up to the required level. ML model cannot fully replace the designing process, although, in this project, we have proven that the ML model can reduce the time required for designing to a huge extent.

8.2 Future Work

Even after many methods to improve the accuracy of models, error in calculating outputs persists. Minimizing or removing those errors is a big task since the circuits behave non-linearly, and their behavior might be difficult to encode such simple machine learning models.

The project's cause can be further pushed by using Machine Learning techniques for design time reduction by helping in technology node change. Circuit topology, once developed, is a difficult task to go through repeatedly for different types of applications. Hence, it can be a fruitful usage of this method to design in one process node and convert the circuit into various other process nodes.

8.3 Project Resources

The project resources are compiled and can be found at GitHub Link

Appendix A

Final Program Flowchart

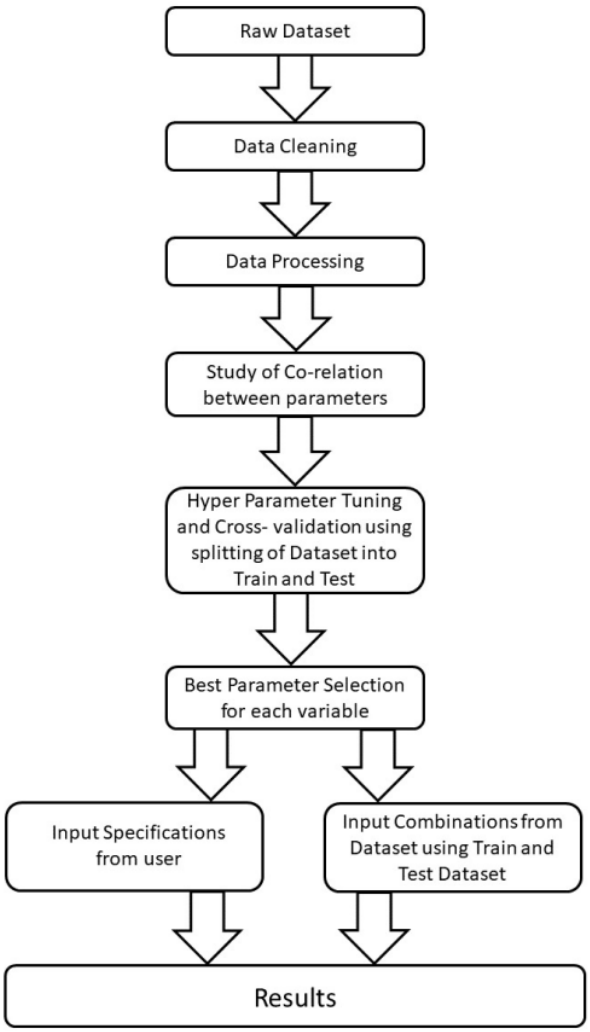

Figure A.1: Flowchart of designed ML Tool

MTP2_Gourav_final_version

ORIGINALITY REPORT

2%	1%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	rei.iteso.mx Internet Source	<1%
2	"Advances of Science and Technology", Springer Science and Business Media LLC, 2022 Publication	<1%
3	Submitted to Coventry University Student Paper	<1%
4	Submitted to University of Southampton Student Paper	<1%
5	Mitsunori Ozaki, Satoshi Yagitani, Hirotsugu Kojima, Ken Takahashi, Akio Kitagawa. "Current-Sensitive CMOS Preamplifier for Investigating Space Plasma Waves by Magnetic Search Coils", IEEE Sensors Journal, 2014 Publication	<1%
6	Anders Forsgren, Philip E. Gill, Joshua D. Griffin. "Iterative Solution of Augmented	<1%

Systems Arising in Interior Methods", SIAM Journal on Optimization, 2007
Publication

7	fr.coursera.org Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

MTP2_Gourav_final_version

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49