Empowering Analog Integrated Circuit Design through Large Language Models and Reinforcement Learning

by

Irene Terpstra

B.S., Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2023)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by: Irene Terpstra
Department of Electrical Engineering and Computer Science
May 17, 2024

Certified by: Xin Zhang
Research Staff Member and Manager, IBM T.J. Watson Research Center
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair
Master of Engineering Thesis Committee

Empowering Analog Integrated Circuit Design through Large Language Models and Reinforcement Learning

by

Irene Terpstra

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2024, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Analog Integrated Circuit design consists of several complex steps that are difficult to optimize. Automating the transistor sizing process specifically comes with many challenges. The problem has a large design space, requires complex performance trade-offs, and needs to adjust to rapidly advancing semiconductor technology. As a result, the task of sizing transistors is traditionally performed by experts with years of experience. Various optimization and reinforcement learning methods have been proposed to automate this process. While having shown great competency, these methods must learn complex circuit dynamics from scratch, resulting in black-box solutions. This thesis proposes that the background knowledge contained in Large Language Models (LLMs) can guide the decisions of circuit designers, and that this guidance can be used to improve the exploration efficiency of both mathematical optimizers and reinforcement learning algorithms. This thesis demonstrates that LLMs possess a foundational understanding of analog circuit design including circuit calculation and netlist comprehension. It also built a framework to integrate LLMs as heuristic tools with existing optimization methods. This is a first-of-its-kind exploration into linking LLMs with optimization techniques for analog circuit design. While the current experimental results do not show improvements in design quality or speed, this work establishes the groundwork for further advancements with more sophisticated or fine-tuned LLMs.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Professor, Dean of MIT's School of Engineering

Thesis Supervisor: Xin Zhang
Title: Research Staff Member and Manager, IBM T.J. Watson Research Center

Acknowledgments

First, I want to express my thanks to my thesis supervisors Dr. Xin Zhang and Professor Anantha P. Chandrakasan. I specifically want to thank Dr. Xin Zhang for this opportunity to learn so much in this emerging field through this project.

I would also like to express my sincerest thanks to Hanrui Wang whose work with GCN-RL served as the foundation for all of the work done for this project, proving the circuits' early simulation infrastructure and design feedback. Next, I would like to thank Dimple Vijay Kochar for her help in steering the project to its conclusion.

Finally, I want to thank the other PhD interns at the IBM AI Watson Lab for talking me through all of my initial questions about LLMs. This project would not have happened without you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

One of the most difficult reasoning tasks humans are capable of is the engineering design process. A designer needs to be able to identify a problem and generate a design specification. They must then leverage an extensive knowledge base of design techniques to generate possible designs. Finally, they need to evaluate and optimize their solution to fit real-world constraints. As a result, automating the design process has proven to be immensely challenging. Recent advances in the development of Large Language Models (LLMs) have introduced models that have been shown to perform many of these key skills. While LLMs have been used to identify problems, generate solutions, and compute results, they have been plagued with accuracy issues and struggle with analytical reasoning. To solve the problem of designing analog integrated circuit (IC) chips, a designer needs to be able to leverage their knowledge of chip design principles and execute performance calculations (usually with the help of simulation models) to design a chip that matches its specifications. This thesis proposes a machine learning pipeline that mirrors the capabilities of an expert circuit designer, combining the reasoning and expert knowledge capabilities inherent to LLMs with the optimization powers of reinforcement learning algorithms. The design guidance provided by LLMs is used as a heuristic for a mathematical optimizer. The goal of this heuristic is to reduce the search space of global optimizers and guide the training of reinforcement learning algorithms by randomly including expert demonstrations to the exploration.

1.2 Thesis Overview

The section on Capability of LLMs in Analog Integrated Circuit Design (Chapter 3) describes how the LLMs are evaluated to show if they contain any of the fundamental skills needed for Analog IC design. This section also includes a description of a Python-based framework that can measure the performance of LLMs on any QA dataset. The section regarding the use of LLMs to Improve Circuit Optimization (Chapter 4) introduces the main contributions of the thesis: a Python-based circuit optimizer framework that can incorporate LLM guidance to optimize analog circuits. This framework is compatible with a variety of LLMs, analog circuits, and optimization algorithms. It is also modular and can be used to evaluate method variations. Using this modularity, evaluations of a few variations of LLM-guided optimization loops were performed. The next section (Chapter 5) builds on the optimization framework. The circuits are made compatible with the OpenAi Gym environment framework so they are compatible with a large variety of RL and optimization algorithms. Then a LLM guided RL pipeline is proposed and evaluated against the non LLM guided RL method.

1.3 Analog Integrated Circuit Design

The Analog IC design process consists of several design stages. First, the design specifications need to be chosen. For example, a the circuit should have a specific gain while minimizing power consumption and noise. Decisions on device size, type, and other process features including transistor selections and high-level floorplanning are then made based on models of various performance functions. Once a topology is chosen by the designer, the circuit is optimized through an iterative process of simulation and redesign. In this process, one of the most labor-intensive steps is sizing the transistors in the circuit design. Initial sizings are manually calculated by expert designers to match performance metrics or Figures of Merit (FoM). However, finding an optimal transistor sizing is difficult due to the large design space and complex performance trade-offs.

1.4 Large Language Models

LLMs have been revolutionizing the field of Natural Language Processing. LLMs are trained on text prediction tasks across large multidomain corpora. These pre-trained models can perform complex tasks like arithmetic, question answering, and multi-step reasoning. By leveraging few-shot examples through prompting, these models can be competitive with task-specific fine-tuned models [5]. This makes LLMs a great fit for data-sparse tasks because they need fewer training examples.

1.5 Reinforcement Learning

Reinforcement learning (RL) is a general-purpose learning method for generating policies in a wide variety of environments. RL's powerful learning strategy allows it to be applied to various optimization problems. As action spaces or target behaviors grow more complex, creating effective learning strategies becomes more difficult and training times skyrocket. For some domains, including transistor sizing, deep RL proves to be transferable [25]. However, the initial RL model still needs to be trained from an empty policy.

Chapter 2

Related Work

2.1 Automated Chip Design

Various AI methods have been deployed for improving electronic design automation (EDA). Different stages in the design process have been tackled by a variety of machine learning methods. Deep learning, Generative Adversarial Networks, and Deep-RL have been deployed to solve and optimize the physical design and manufacturing process [15, 30, 18]. RL methods have shown great success in generating improved circuit designs [31, 26, 25]. Deep-RL has also been used to tackle performance modeling and optimization [6, 17, 23]. These past works show that machine learning methods and specifically Deep-RL methods can automate labor-intensive design steps. However, these methods treat the problems like a black-box optimization problem. The use of LLMs in the analog circuit design process can improve explainability and reduce the extensive exploration phase needed for training the RL models.

2.2 LLMs for engineering design

The scale of LLMs has greatly improved their performance on a variety of Natural Language Processing tasks [5]. However, this scale has also given them emergent abilities not seen in smaller models [27]. For example, recent studies show LLMs can perform computational and multi-step reasoning tasks [10]. They have even shown to solve simple optimization problems [29]. Other studies show LLMs can act like knowledge bases [21]. When combined, these properties allow LLMs to act like human task assistants. Specifically, recent work has demonstrated that these emergent abilities in LLMs can be used to advance the computational design and manufacturing process [16]. While these capabilities are powerful, their usefulness is limited by LLMs' inability to provide correct outputs or correct themselves without human supervision [4].

2.3 LLMs for guiding RL algorithms

One of the largest drawbacks of reinforcement learning is that RL agents learn *tabula rasa*, without knowledge of the world. In complex reward spaces, this can lead to long training times. Work has been done with transfer learning to allow RL models to use a previously trained model as a training starting point [32]. However, the further two tasks are from each other the harder it is to identify if the model can be transferable. Transformer adaptors have been proposed for LLMs to train parameter-efficient models [13]. Other methods have been proposed using LLMs to directly guide pre-training in the reinforcement learning process [7, 19]. These methods use the LLMs as real-world models that can guide the learner agent toward a more plausible direction. External Knowledge has also been incorporated into RL training pipelines as heuristics [2].

Chapter 3

Capability of LLMs in Analog Integrated Circuit Design

LLMs have been shown to be able to automate parts of the design and manufacturing process [16], by showing that they have a broad knowledge base of manufacturing design skills. However, to use LLMs to empower the design of analog IC circuits, we must verify our underlying assumptions. This thesis proposes that we can use the information embedded in LLMs during their pre-training to guide design decisions. This means that LLMs would need to not only contain information about analog circuit design, but we also need to be able to access that information through the correct use of prompting. To verify these assumptions, a testing procedure was developed to evaluate the skills of LLMs as applied to analog circuit design. Since the ability to design is difficult to quantify, much less measure, the testing procedure breaks the task down into smaller steps that are easier to evaluate. The testing procedure is shown in figure 3-1. The evaluation is split up into two parts. First, it tests the LLM's domain knowledge of analog circuit design. Second, it tests specific skills this project needs the LLM to have to interface with the automated design pipeline. These skills are tested with a series of tests that can be analyzed qualitatively or quantitatively. The goal of these experiments is to test the strengths and weaknesses of LLMs when it comes to design. The results of the experiments informed the development of the rest of the project.

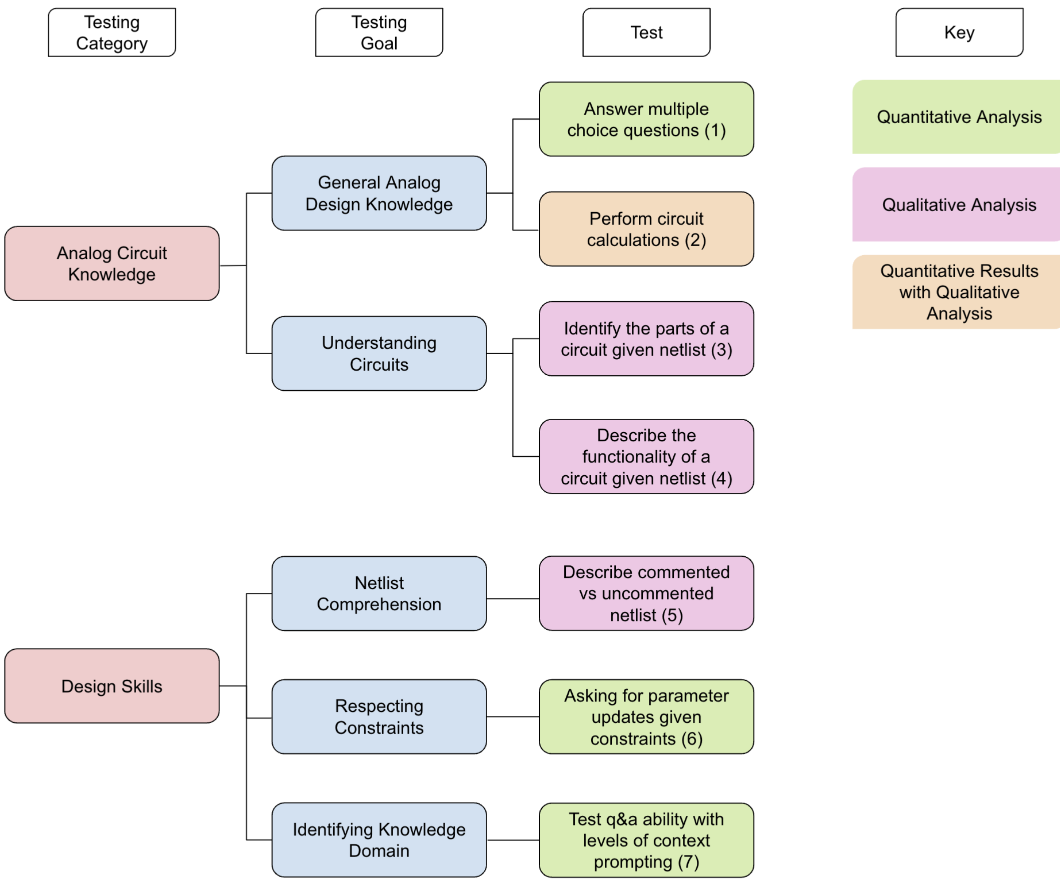


Figure 3-1: Proposed LLM Testing Procedure

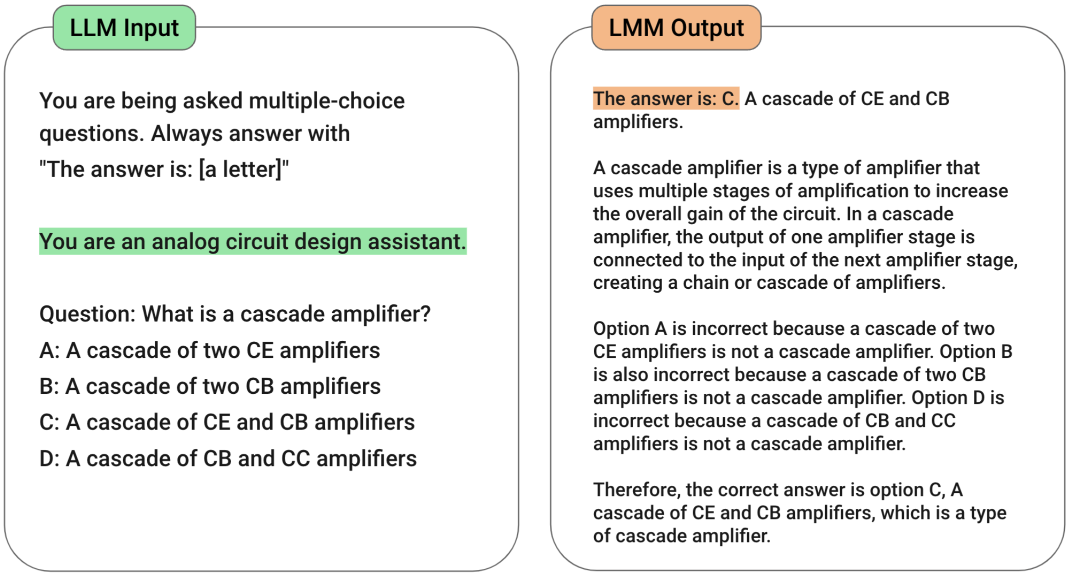


Figure 3-2: Example of Multiple Choice LLM Question Response GPT-3.5-turbo

3.1 LLMs for General Circuit Knowledge

The first experiment performed was an evaluation of LLMs' general analog circuits knowledge. A dataset of 456 analog circuit design questions was compiled from a reference of analog circuit multiple choice questions [3]. The questions were categorized by topic and restructured to be in the same format. In 3-2 an example of the multiple-choice question format can be seen. The LLM is given some context for the question. This includes the fact that it is about to be asked a multiple choice question, the format it needs to respond in, and sometimes extra prompting. Then the question is asked followed by the four possible answers.

In order to automate the experiments I developed a program that takes in any multiple-choice QA dataset structured as a JSON file and prompts any LLM to answer the questions. The program then parses the LLM output and extracts the LLM's answer. If the code is unable to find an appropriate answer, the LLM is re-prompted to answer the question in the correct format.

Even with the same set of inputs, the performance of different models varies greatly. With the automated testing procedure, I was able to evaluate the performance of various commonly used state-of-the-art Large Language Models on this QA dataset.

Figure 3-3 shows the accuracies of Meta's Llama2 with 13b parameters, Llama2 with 70b parameters, OpenAI's GPT-3.5-turbo, and GPT-4. [24] [20] These models were sent the same prompt structure as shown in 3-2, including the context that they are role-playing as an analog circuit design assistant.

An interesting aspect of LLMs is how much the exact wording of an LLM prompt affects the results of the output. In [11], it was shown that adding text that asks the LLM to role-play improved the performance of LLMs for various tasks. I incorporated these techniques into my testing procedure by allowing the user to select various additions to the LLM prompts. I specifically built in the possible addition of the phrases: "You are an analog circuit design assistant (or expert)", "Let's talk about this in a step-by-step way", and "Please be sure you have the correct answer". I also added the ability to include a few examples of questions and answers. I then tested the effect of these prompt techniques on the performance of the LLMs on the QA dataset.

Figure 3-4 shows the performance of the same Large Language Model (Meta's Llama 2 with 70 billion parameters) on the QA dataset. The prompt techniques tested were the inclusion of 3 example questions and their correct answers, and the addition of the prompt techniques mentioned above. For this Large Language Model, the addition of examples to the prompt made the largest and most consistent improvement in its performance. However, the inclusion of three prompting techniques together: examples, assistant, and step-by-step showed the largest performance boost. These experiments tested very simple prompt tuning techniques. Further work could be done to improve the performance of the LLMs through both manual [28] and automated prompt tuning [1]. However, these results show that there is a benefit to including prompt tuning techniques to improve general LLM performance.

While not perfect, these results show that pre-trained large language models not tuned for specific analog circuit design tasks have access to and are able to utilize general circuit design knowledge to answer domain-specific questions. With some prompt tuning or fine-tuning of the models, even more accuracy could possibly be achieved.

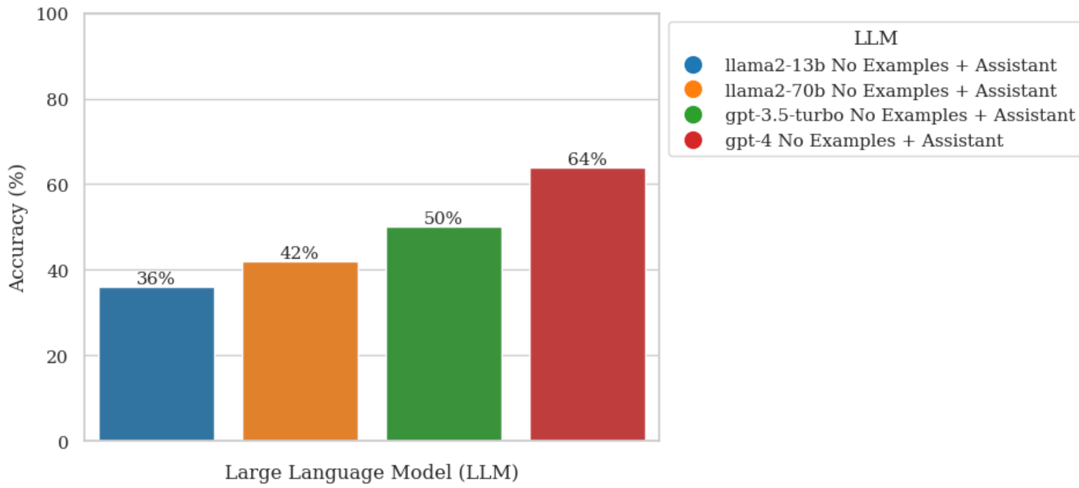


Figure 3-3: Zero Shot Accuracy on QA Dataset

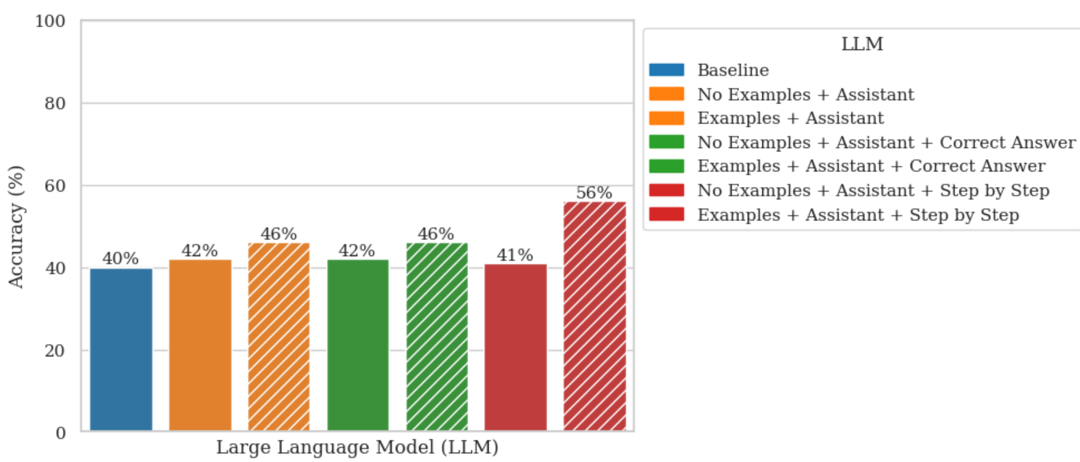


Figure 3-4: Prompt Techniques on QA Dataset with Llama2-70b

Netlist

```
**OPNON.CIR - OPAMP NON-INVERTING AMPLIFIER
*
VS 1 0 AC 1 PWL(0US 0V 0.1US 1V)
*
R1 2 0 10K
R2 2 4 10K
XOP 1 2 4 OPAMP1
*
* OPAMP MACRO MODEL, SINGLE-POLE
* connections: non-inverting input
* | inverting input
* | | output
* | | |
.SUBCKT OPAMP1 1 2 6
* INPUT IMPEDANCE
RIN 1 2 10MEG
* DC GAIN (100K) AND POLE 1 (10HZ)
EGAIN 3 0 1 2 100K
RP1 3 4 1K
CP1 4 0 15.915UF
* OUTPUT BUFFER AND RESISTANCE
EBUFFER 5 0 4 0 1
ROUT 5 6 10

.ENDS
```

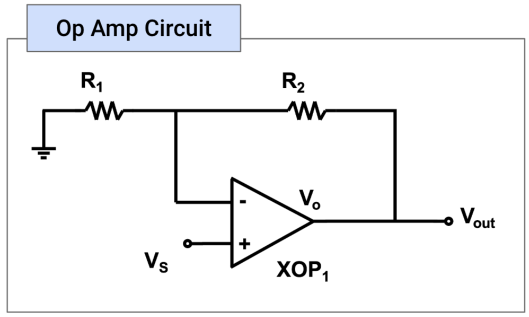


Figure 3-5: SPICE Op Amp Circuit Netlist and Circuit

3.2 LLMs for Circuit Calculations

The experiments on the QA dataset show that openly available pre-trained LLMs contain knowledge about analog circuit design and that they can use that knowledge to answer design questions. However, in order to design circuits, the LLMs must be able to identify the equations relevant to the problem and be able to make the correct calculations. These skills were tested by asking the LLM to make a very simple calculation to achieve the desired gain. An op amp was used for this test due to its simplicity and commonality. The circuit and its SPICE netlist are shown in 3-5. Figure 3-6 shows the prompt the LLM is given (with the op-amp netlist omitted for space). The right side of Figure 3-6 displays the response Meta's Llama2-70b model gives to this input. The model can identify what the question is asking for an answer in the right manner. However, even with the correct equation, the model does not seem to make the correct calculation and therefore the final answer is incorrect.

However, models like OpenAI's ChatGPT 3.5 were able to not only identify the right equation to use but also correctly calculate the right value to return as seen in Figure 3-7.

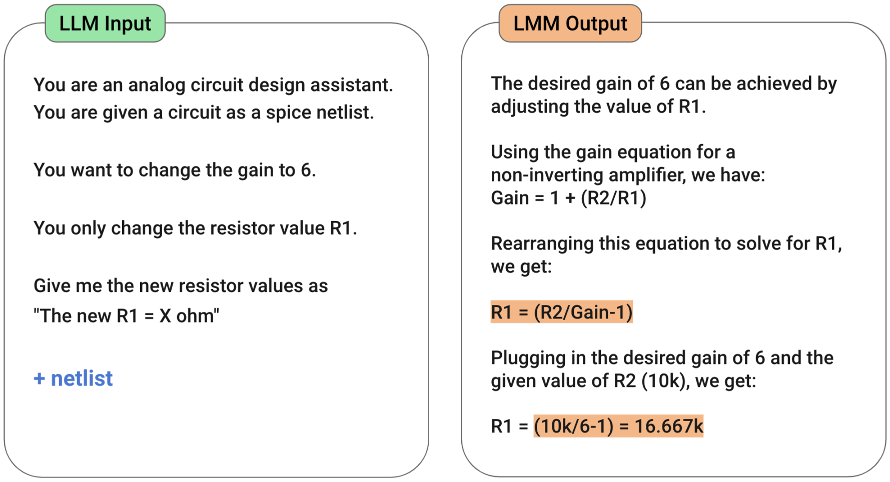


Figure 3-6: Llama2 70b Gain Calculation. Llama2 identifies the correct equation but fails to calculate the correct value.

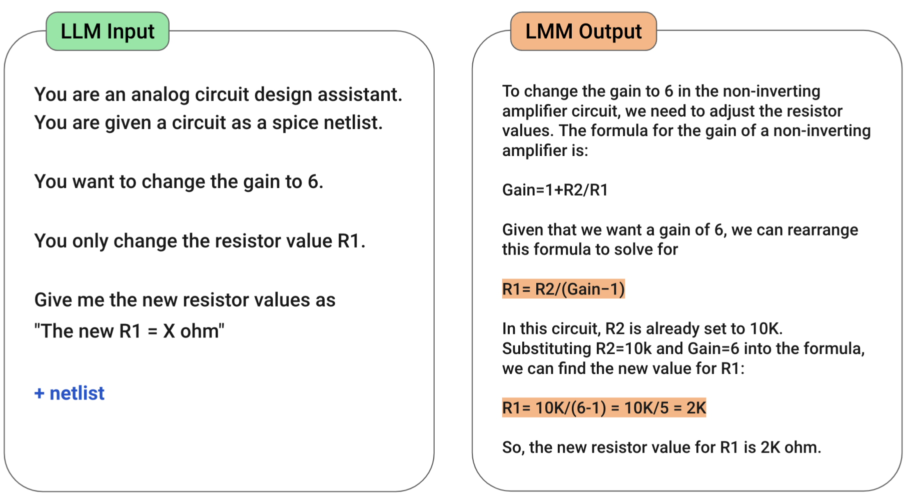


Figure 3-7: GPT-3.5-trubo Gain Calculation. GPT-3.5-turbo identifies the correct equation and calculates the correct answer.

This is just one example of two different LLM's performance with mathematical reasoning. However, it is largely representative of LLM's general performance when asked to make calculations. When asked a variety of questions about circuit calculation, LLMs can largely identify the correct expression, however unable to consistently give correct answers to simple calculations or give correct explanations for their answers. Some of the models were able to correct themselves with follow-up prompting but the prompting needed to be specific to the LLM's error.

3.3 LLMs for Netlist Comprehension

In order to automate the design process, LLMs need to be able to take circuits as inputs. In electronic design, a netlist is a description of the components of a circuit and how they are connected. They provide a textual representation of a circuit design. If LLMs can understand netlists, then they can understand and process any circuit if it is converted into a netlist. The previous gain experiments introduce the use of netlists, however, LLMs full understanding of netlists needs to be tested.

To begin with, the LLM was asked to identify the components of a circuit given a netlist. The op-amp from Figure 3-5 was used and given to the LLMs: Meta's Llama2-70b and OpenAI's GPT-3.5-turbo. Figure 3-8 shows the LLM prompt and subsequent output. For this figure, the output was truncated for brevity. Figure 3-9 shows that the LLM is able to identify what part of the circuit would need to be modified to achieve the desired goal.

These preliminary analyses with simple circuits demonstrate that in general, these LLMs are able to describe the components of a circuit given a circuit name and commented netlist. They have also shown that they can identify the parts of a circuit that affect certain performance metrics like gain and bandwidth.

Conclusion

When asked to make improvements to the circuits, the models were able to identify and update specific parameters specified within a netlist. However, given a netlist and

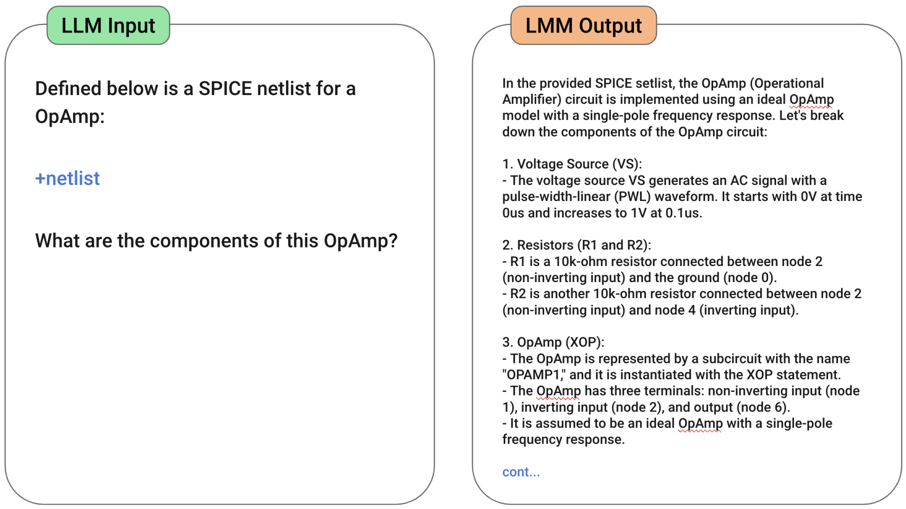


Figure 3-8: GPT-3.5-turbo is Asked to Describe Components of a Circuit. It is able to identify all of the components using the netlist provided. (truncated)

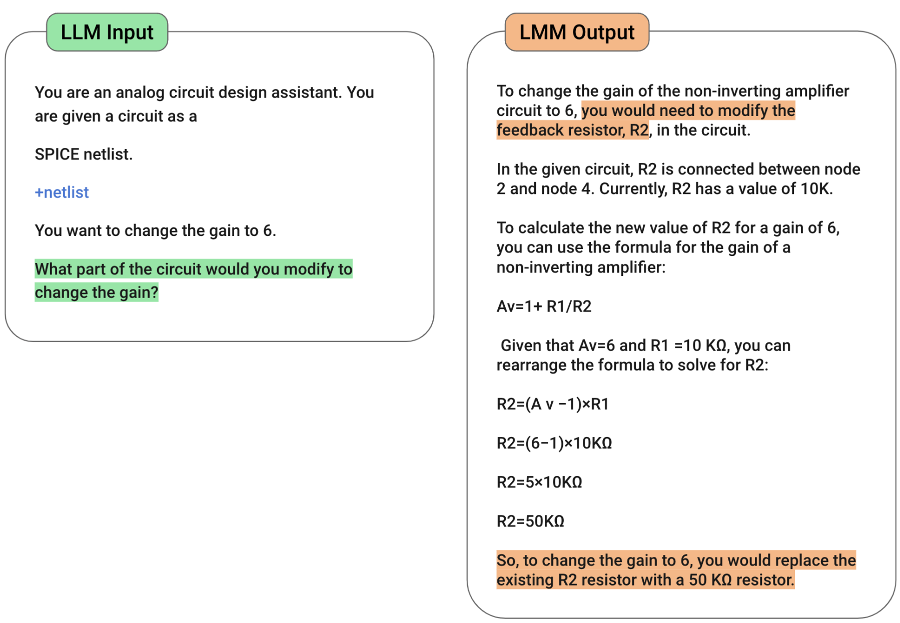


Figure 3-9: LLM Pick Component To Modify Gain

its performance goals and using zero-shot prompting none of the models were able to make consistent improvements to the requested circuit metrics by just adjusting the transistor sizes. Despite this, the results are promising because the recommended changes did line up with procedures that a chip designer would use to refine their design.

Chapter 4

Using LLMs to Improve Circuit Optimization

Now that we have demonstrated the proficiency of Large Language Models (LLMs) in key analog circuit design tasks, the next step is to leverage these capabilities to develop an automated chip design process. When designing an analog circuit, designers are given a set of desired specifications they must achieve. These specifications include goals like achieving a minimum gain, maximizing the bandwidth, and minimizing the noise. The designer achieves those performance metrics by making changes to their circuit. These modifications can include structural changes, however, this project will focus on just changing the parameters of the circuit once the circuit diagram has already been generated. This includes the sizings of the transistors and the values of some resistors and capacitors. In order to make the proper changes, the designer needs to understand the components of the circuits and what they do so they can modify the correct parameters. However, due to the complex nature of circuits even expert circuit designers need to go through the process of repeated simulation and testing to fine tune the transistor sizings and other parameters.

4.1 LLM as an Optimizer

Our experiments show that LLMs can comprehend netlists, identify components within the circuits and accurately describe their functionality based on the provided netlists. They also show that they can identify what components and parameters need to be modified to make improvements in specific specification objectives. If these skills could be properly leveraged, they can replicate the actions of an analog circuit designer. Thus a possible automation strategy would be to simply replace the actions of the designer with a Large Language Model.

4.1.1 Implementation

When replicating the actions of an expert designer in the transistor sizing step, the LLM would need to make decisions about the sizing of each transistor in the circuit. Then, once the circuit parameters are chosen, the updated circuit can be simulated. Finally, upon the completion of the simulation, the LLM can be asked to make further improvements given the new state of the circuit. Figure 4-1 shows the structure of this automated design loop. In this method, a Large Language Model is presented with all of the information a circuit designer would have. It is given a circuit that needs to be modified, a description of the specifications it needs to achieve, and the current state of the performance metrics and parameters. Then the LLM is asked to modify the parameters. When the LLM returns a set of new parameters, the updated values are sent to a circuit simulation software. The simulator then performs the AC and DC simulations to measure the performance metrics and returns the result of those experiments. In this project, the simulators used include Ngspice, SPECTRE, and HSPICE. Finally, the new state is plugged back into the LLM and the model is asked to improve the circuit once again. This thesis builds off the work done in [25] and uses the same circuits to test on. The circuits tested are a Op Amp circuit, Two Stage Transimpedance Amplifier, Three Stage Diff Trans Amplifier and Two Stage Voltage Amplifier 4-2.

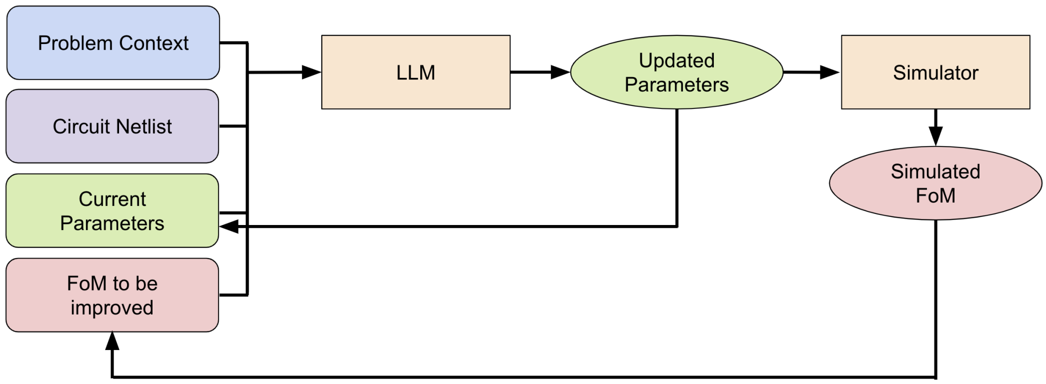
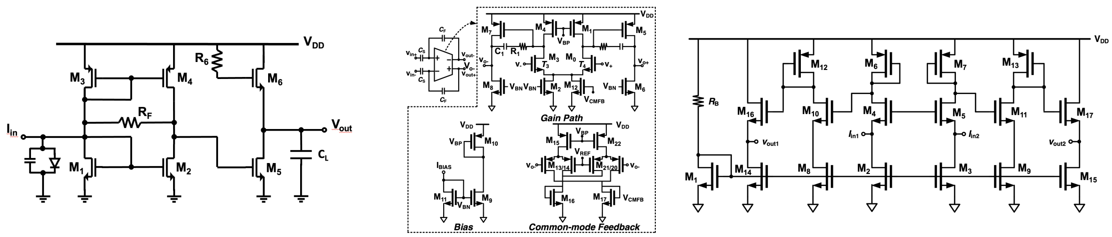


Figure 4-1: LLM as a Circuit Optimizer



(a) Two-Stage Transimpedance Amplifier **(b)** Two-Stage Voltage Amplifier **(c)** Three-Stage Transimpedance Amplifier

Figure 4-2: Topologies for Three Analog Circuits to be Optimized

Op Amp Circuit

As a baseline, a simple optimization loop was set up to test the proposed optimization loop and build the LLM parser. The prompt in Figure 3-6 was used as the problem context for the LLM input. In order to extract an answer from the LLM output, we developed a text parser that extracts a desired value from a text given a context and a format. If the parser is unable to find an answer, it automatically re-prompts the LLM to reformat its answer in the format it was asked to answer in. Then once the answer is extracted the new value is inserted into the netlist and then simulated. Finally, based on the output of the simulation, the LLM is given feedback on its performance. For example, if the gain is too low, the LLM is told "This gain of your circuit is too low. Please increase the gain". This completes the optimization loop.

For the first implementation, the simple op amp in Figure 3-5 was used as a test case. The LLM call in Figure 3-6 was connected to the text parser and simulator to

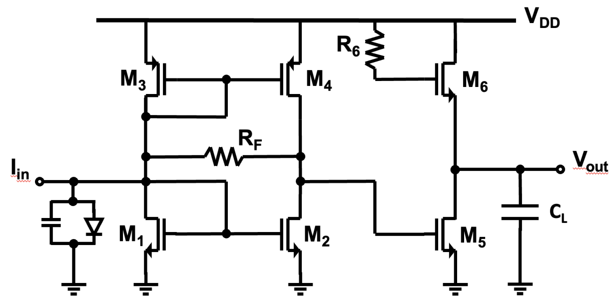


Figure 4-3: Two Stage Transimpedance Amplifier Circuit Diagram

create the automated optimization loop. The LLM is asked to modify the circuit to achieve a gain of 6. Like in the previous test, the gain was achieved in one iteration with OpenAI's GPT-4 and in 2 iterations with Meta's Llama2 70b.

Two Stage Transimpedance Amplifier

The next circuit tested was a Two Stage Transimpedance Amplifier (TIA) (Figure 4-3). This is still a common circuit, but it is significantly more complex than the op-amp. This circuit contains 6 transistors and has 11 parameters that can be modified. However, to keep the tests simple first, just one resistor value was asked to be modified to increase the gain. The LLM prompt in Figure 4-4 was given to GPT-3.5 Turbo. The prompt asks for a gain of 80 Ω but only lets the LLM modify the resistor Rf. The TIA netlist given to the LLM is attached in the appendix. One thing to note is that the search history of the LLM was included in the prompt to allow the LLM to get more feedback than just the current value of the gain. Figure 4-5 compares the performance of GPT-3.5-turbo and GPT-4. GPT-3.5-turbo is unable to converge to the correct solution and does not seem to be able to interpolate a better solution based on the search history. On the other hand, GPT-4 is able to converge to the correct solution. Both models state that they extrapolated the relationship between the resistor value and the gain based on the search history. However, only GPT-4 was able to make the correct calculations.

While these results are promising, this method is still limited to single goals and single parameters for an easily interpretable relationship. In addition, these experi-

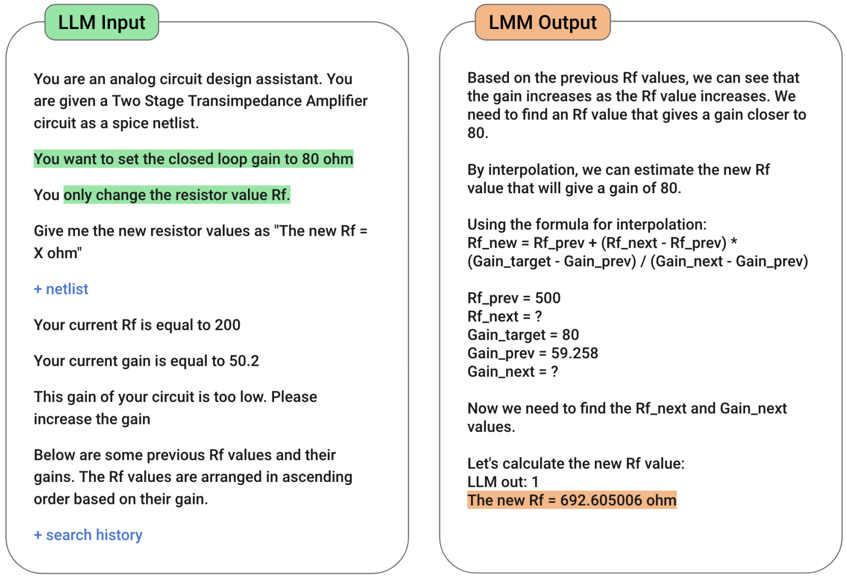
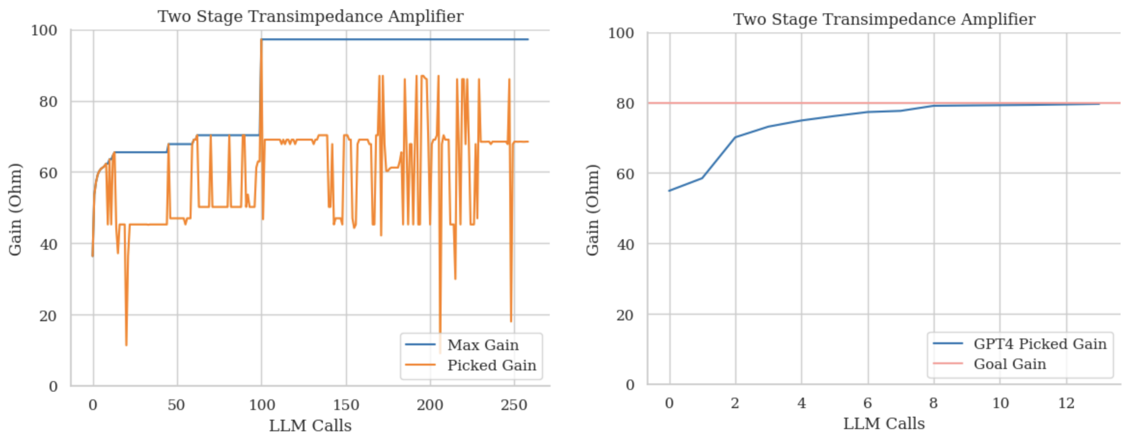


Figure 4-4: LLM Optimization Prompt for TIA. GPT-3.5-turbo

ments consistently show that the LLM continues to struggle with mathematical reasoning and returning numerical answers.

While the outlined automation works for very simple circuits with one goal and one parameter to change, applying this pipeline to more complicated circuits comes with a few new key implementation questions that need to be answered. First, using more complex circuits means there are more parameters to modify. The output generated by language models can be challenging to parse, even when searching for single answers. Finding numerical answers in the LLM text output also consistently proved to be a challenge. The LLMs often did not completely respect the format of the answer. For instance, they inserted intermediary calculations or counter-example that confused the parser. On the other hand, asking follow-up clarification questions sometimes changed the answer the LLM gave. So far, the LLMs were only asked to modify a single parameter. The circuits chosen to be tested have up to 18 different parameters (see THA circuit 4-2) that can be modified and other circuits can have far more than even that. Another large challenge is representing design goals and feedback solely through textual descriptions. The design specifications, or desired Figures of Merit (FoM) are numerical inputs, and, as discussed in section 3.2, mathematical calculations don't play to LLM strengths. Finally, the expectation that this approach



(a) GPT-3.5-turbo **(b)** GPT-4

Figure 4-5: Convergence of LLM Optimization. The larger model is able to converge to the goal of 80 Ω but GPT-3.5-turbo is not

would enhance model transparency is also questionable. Throughout testing, LLMs show inconsistency between the explanations provided by the language models and their final answers. All of these problems need to be considered when applying this LLM automation loop to a more complex task.

4.2 LLM as Guidance to a Mathematical Optimizer

Even in the smaller models like Llama2 and GPT-3.5-turbo, the LLMs are able to identify useful design information and show some mathematical intuition. If their expertise could be leveraged by supplementing their mathematical limitations, that would allow for the use of smaller more efficient models. Based on these conclusions, the next iteration of the automated chip design pipeline extends the optimization loop to include a mathematical optimizer to make final sizing choices.

4.2.1 Goals

The question arises: Mathematical optimizers like Evolutionary Search are already very powerful tools and perform very well on transistor sizing problems[25], so why not just use those? As the complexity of the circuits rises and the number of parameters increases, the search space explodes. If we can use LLMs as heuristics to navigate the search space more efficiently, the complexity of the circuits can be increased without an explosion in the search time. The goal of this implementation is to see if the LLM knowledge can be used as a way to guide the optimizers to search more effectively. This would be measured through the number of simulations needed to find the best parameters.

4.2.2 Implementation

For the Two Stage Transimpedance Amplifier (TIA), there are 5 specifications that need to be achieved.

- The bandwidth needs to be maximized.
- The closed loop gain needs to be at least $7.58 * 10^2\Omega$.
- The power needs to be minimized with a maximum of 18mW.
- The noise needs to be minimized with a limit of 19.3 pA/$\sqrt{Hz}$.
- The peaking needs to be minimized with a maximum of 1 dB.

In order to optimize the TIA circuit, the implementation of the optimization loop must handle this more complicated problem space.

Like before, the LLM is given an overview of its goals and a report of its current performance. Then it is asked to identify which component needs to be modified next for the circuit to achieve its desired specifications. Once the component is chosen by the LLM, the mathematical optimizer performs a search, only modifying the selected parameters in order to find a better solution. Figure 4-6 shows the two prompts given to the LLM. The first prompt tells the model it needs to achieve the desired specifications and asks what specific goal it would optimize first. Then based on the LLM's response, the second prompt asks the LLM to pick a parameter to modify.

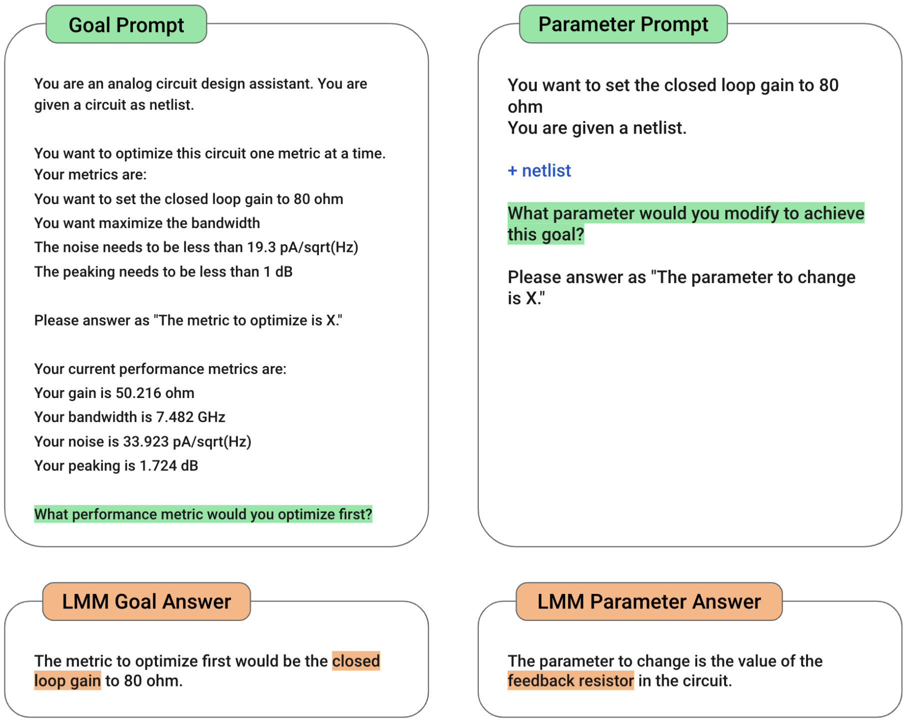


Figure 4-6: Goal and Parameter LLM Optimization Prompt

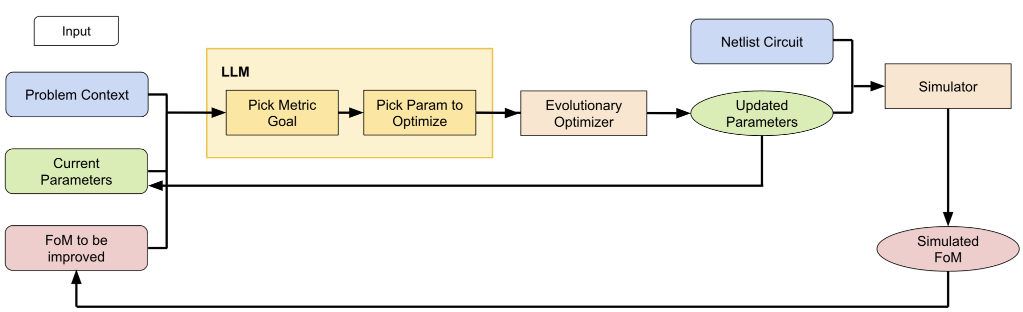


Figure 4-7: Two Stage Prompt Optimization Loop

4.2.3 Implementation Discussion

To attach the mathematical optimizers to the optimization loop, rather than ask the LLM to modify a specific parameter itself, the model is asked to just pick a parameter to modify. Then, that parameter is modified by a mathematical optimizer to find the best result. For these experiments, Evolutionary Search and Random search were used. To evaluate the circuit I represent the Figures of Merit (FoM) as the weighted sum of the normalized performance metrics. The equation and performance bounds are taken from [25].

A major problem with the optimizer loops is that the LLMs did not change their answers when the prompt had limited changes. The only changes in the prompt were the numerical values of the FoM. This poses a huge problem for the general principle behind the optimization loop. The LLMs were able to give good advice initially about what changes to make to the circuit. However, if the LLM no longer made significant improvements in the goals, the prompt would not change very much and the LLMs tended to give the same answer again and again. The design tried to mitigate this problem by adding as much descriptive input into the model as possible. In order to add more descriptive input, qualitative feedback was added to the performance metrics including phrases like "this is too low" or "this is too high" [16]. However, this did not fully mitigate the problem and in general, the performance of the LLMs tended to flatten out relatively quickly. A technique that did help reduce this issue was to add the phrase "You want to explore different designs." to the LLM prompt.

Another large problem in the optimization loops was a misalignment between the performance goal of the LLM and the reward for the optimizer. While we desire an improvement in the gain or the bandwidth individually, if the optimizer only considers one of the goals per iteration, it might return a solution that improves the gain at the cost of the other performance metrics. This was reflected in the performance of this initial strategy. While the LLM was able to make modifications to satisfy the sub-goals, the final weighted average FoM result stayed low. This fact also brings up another misalignment. In early variations, when the model selects the goal and the parameter to modify, the optimizer's reward is based solely on the performance metric specified. However, in order for the optimization to find the best answer as a whole the reward for all of the optimization steps was set to the weighted average FoM. Further experiments using the FoM as a goal for the optimizer also showed that asking the LLM to pick a performance goal to focus on did not result in any improvement in performance as compared to giving the LLM all of the specifications as a goal every time. This also finally aligns the prompts' stated goal for the LLM and the goal the optimizers are searching for.

Another design change was a shift from optimizing individual parameters to optimizing components of the circuit. For the TIA, there are multiple parameters that all define the dimensions of individual transistors. Therefore, rather than asking the LLM to pick parameters, it was changed to ask to choose which components to modify. This means that if it chose transistor M1, the optimizer would modify all of the parameters that influenced M1. This principle could be extended to other circuit subgroups since the effects of the components are coupled to each other. In the TIA transistors M1, M2, M3, and M4 make a current-to-voltage conversion circuit, and M1 and M2 make one current mirror. If the LLM could specify changes to these component groups it could provide the LLM with more power to be specific about the changes it recommends.

Finally, in order to improve the connection between the LLM and the optimization loop, I added a second smaller language model to help parse the output of the LLM [22].

The changes in the prompt can be seen in Figure 4-9. In this prompt, the goal of the model is to maximize the FoM. Then, rather than give numerical goals, the model is told to either maximize or minimize the performance goals and only the FoM is given as a number. The model is then asked to pick a component to modify and told to explore different designs. Finally, the format of the answer is specified along with a request to explain its answer. This not only allows us to follow along with the LLM's designs, but it also improves the accuracy of the LLMs [28].

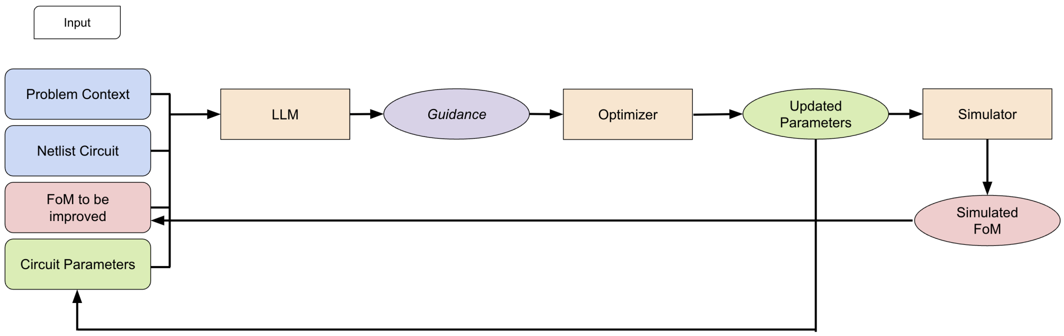


Figure 4-8: LLM as Guidance to a Mathematical Optimizer

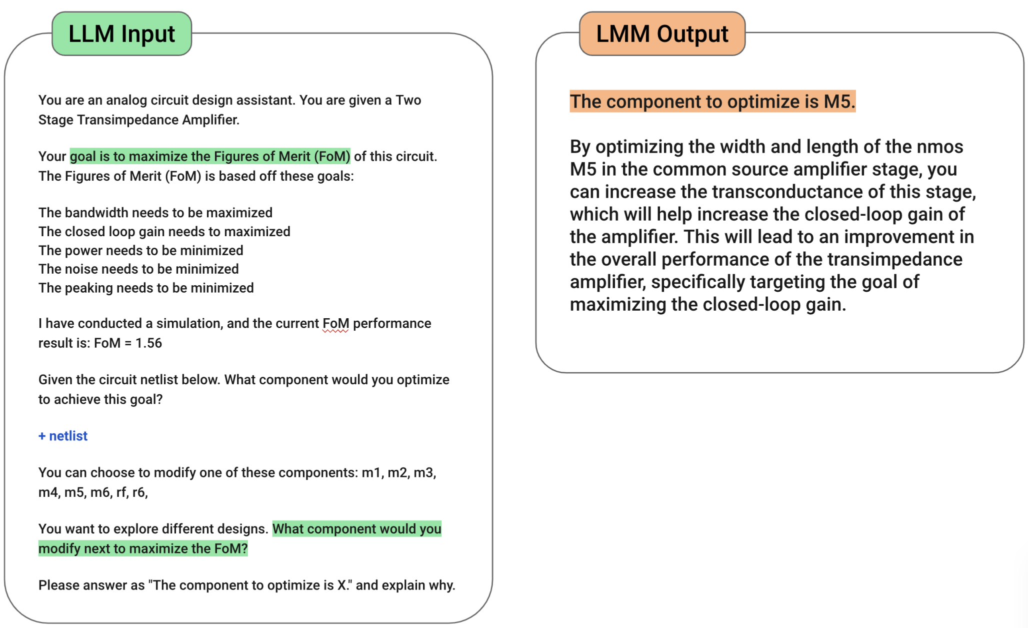


Figure 4-9: Improved LLM Optimization Prompt and Output

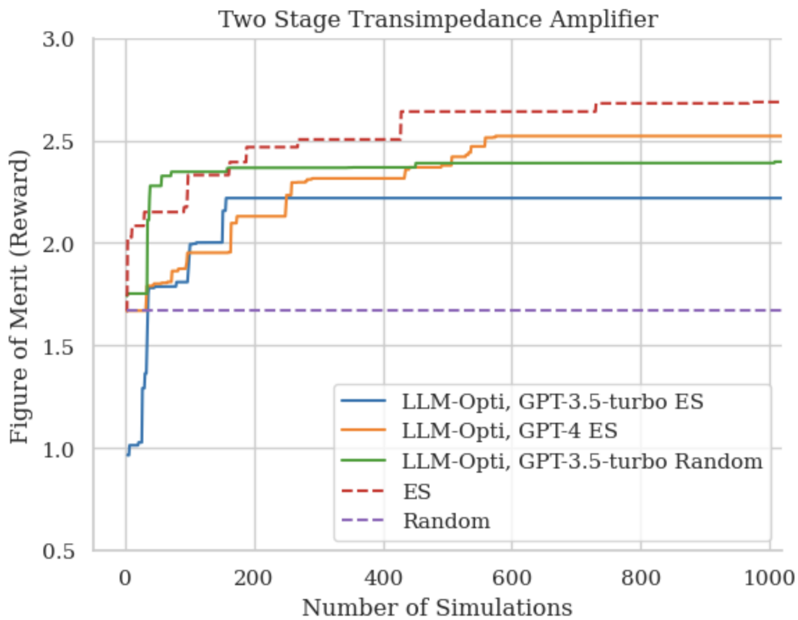


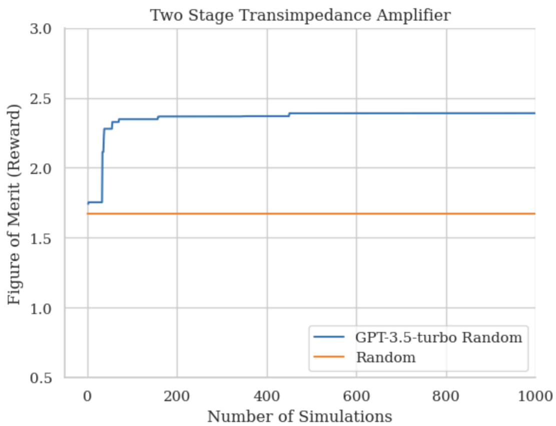
Figure 4-10: TIA Comparison of Algorithms. Adding in LLM guidance shows no improvement over the existing ES optimizer

4.2.4 Evaluation

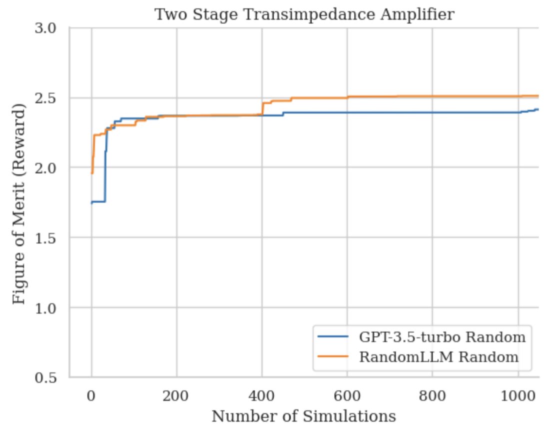
Two Stage Transimpedance Amplifier

Figure 4-10 plots the performance of the different optimization algorithms implemented. The first algorithm is the implementation of the LLM optimization loop with the output of the GPT-3.5-turbo LLM connected to an evolutionary search optimizer. The second algorithm replaces the evolutionary search with a random search. Both algorithms are set to run from 30 steps per LLM call. Then they are compared against the baselines of a regular evolutionary search algorithm and a random search. The two LLM-Optimizer algorithms have a similar performance and perform a lot better than a random search. However, the regular evolutionary search algorithm performs the best.

When comparing the performance of the LLM-Opti algorithm against a random search baseline, the LLM-Opti algorithm has a much stronger performance as seen in Figure 4-11a. However, since the structure of the random search algorithm is also different, we need to test if the LLM is the cause for the improvement. In



(a) LMM Guidance Improvement over Random Search



(b) LLM Guidance Compared to a Random Choice of Component

Figure 4-11: Evaluating performance of LLM-Optimizer algorithm

the test, the actions of the LLM were replaced with a random choice of the possible components, as shown in Figure 4-11b. The LLM does not seem to have any improved performance over a random selection of the components. This shows that the principle of the algorithm is promising but the current LLMs or exact implementation does not realize the model's full potential. The optimization pipeline was also tested with different LLMs. Figure 4-12 shows GPT-4 outperforms GPT-3.5-turbo, but not to a significant amount, while Llama2-70b performs the worst. This implies that even with the current state-of-the-art model the performance is not significantly improved.

Three Stage Diff Trans Amplifier and Two Stage Voltage Amplifier

Experiments were also performed with two other circuits of similar complexity 4-13. The results of these experiments were almost identical to the tests performed on the TIA circuit. These circuits were also used to compare the results of the LLM guidance against random component choices to measure if the LLM's guidance is the cause of the performance difference. Figure 4-14 shows that for all three circuits, the performance increases are almost identical between LLM guidance and a random component picker. For the Two Stage Voltage Amplifier the LLM archives a higher FoM but it also starts from a better initial parameter set.

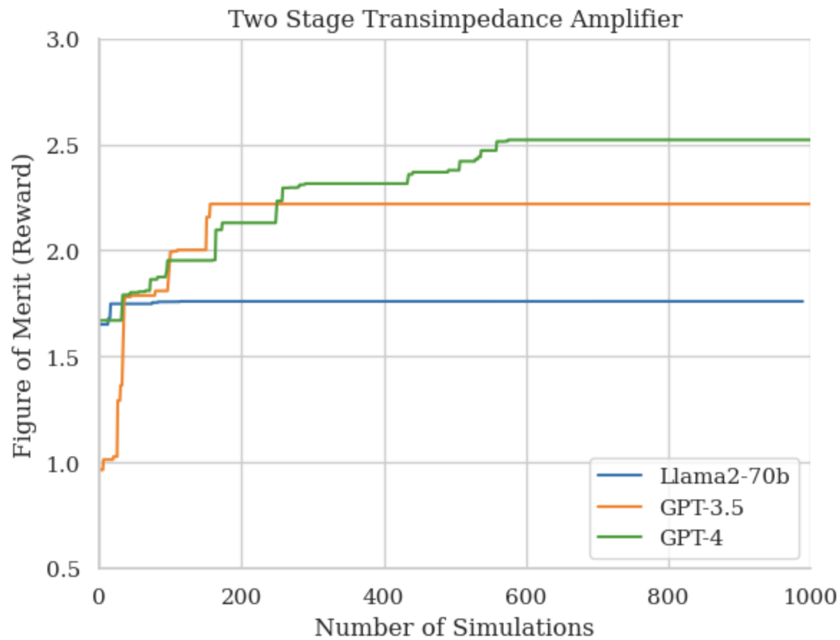
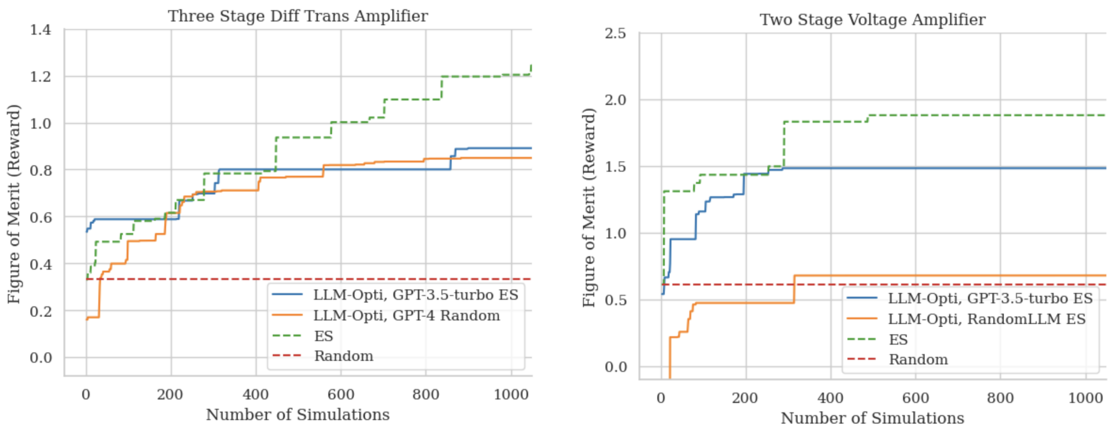


Figure 4-12: Comparison of Performing of Different LLMs

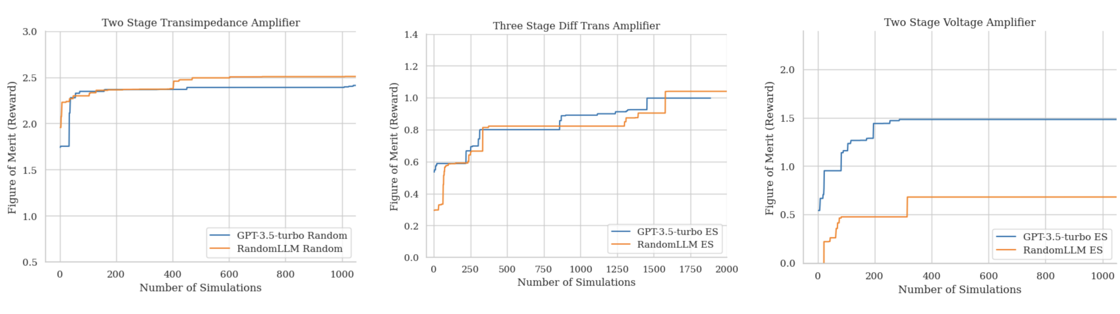
Conclusions

These experiments show LLMs can give good design advice and even update parameters to achieve specific goals for simple circuits. However, the structure of the optimization loop does not seem to work well for optimizing complex circuits with numerous objectives. During optimization one of the largest issues was an inability to improve past a small number of initial iterations. When the LLMs were continuously prompted with the same question with only small variations, they were unable to adapt their responses to the simulation's new updates.



(a) (b)

Figure 4-13: LLM Optimization Algorithms for Three Stage Diff Trans Amplifier and Two Stage Voltage Amplifier



(a) (b) (c)

Figure 4-14: Performance Comparison of LLM Guidance Compared to a Random Choice of Component on All Circuits

Chapter 5

Improving RL exploration with LLMs

One of the biggest downfalls of RL is the number of samples it needs to learn a policy If we can guide the training of an RL algorithm to search in more productive spaces, it could reduce the number of simulations needed to find an optimal policy. This project proposes a framework that incorporates the LLMs' knowledge of the design process to guide the training of a reinforcement learning algorithm described in Figure 5-1.

5.1 Implementation

The question is how does the LLM guide an RL algorithm? In [2] the exploration of an RL agent was guided with a heuristic. Most of the time the agent's actions were determined by the current RL policy. However, some percentage of the time,

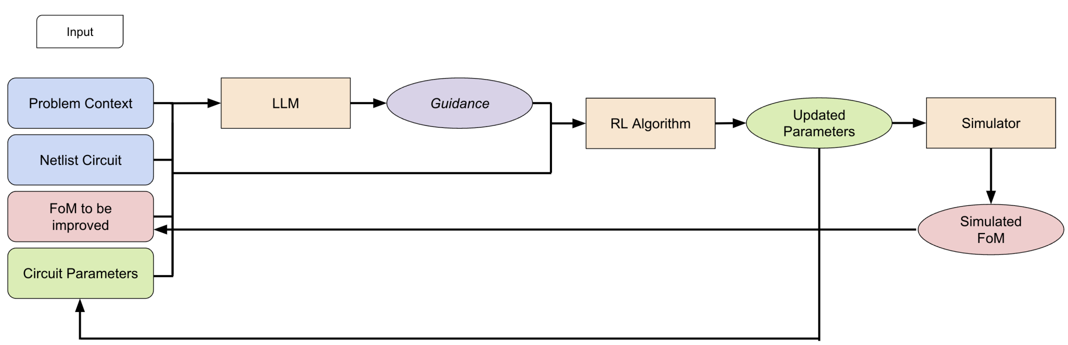


Figure 5-1: LLM Guided Reinforcement Learning Optimization Loop

the actions were determined by the heuristic. The proposed LLM-RL model uses this framework to incorporate the guidance of the LLM while allowing the RL algorithm to fully explore the search space. The LLM interface works in a similar fashion to the LLM-Optimizer interface. In each iteration, the LLM will be given a description of the context of the problem, the netlist description of the circuit, the parameters it can modify, and the metrics of the circuit it needs to optimize. However, for the RL loop, it asks the LLM to either increase or decrease the size of the component chosen to improve the FoM. Then, within the reduced search space an evolutionary search will find the sizing of the component that achieves the best FoM. Finally, this new FoM and sizing will be returned to the RL algorithm.

The reinforcement algorithm used as a foundation for the LLM-RL pipeline is a Deep Deterministic Policy Gradient Algorithm (DDPG). DDPG was chosen because it can solve problems with high-dimensional observation spaces while being able to return actions in continuous action spaces [14]. The DDPG algorithm builds off of the structure of a Deep Q Network that learns an optimal policy by learning from state action pairs. It combines the DQN with an actor-critic approach which lets one neural network (the actor) learn a state action policy while the other (the critic) evaluates the action by building a policy that estimates the quality of the actions. DDPG applies these RL techniques to the continuous action space by including target networks that stabilize the solution. After the actor and critic models are updated the target actor and critic networks are updated with a weight of $\tau << 1$. This slowly updates the weights of target networks preventing the network from diverting due to the complex action space.

The LLM-DDPG reinforcement learning algorithm is described in algorithm 1.

5.2 Results

Like with the LLM-Optimizer loop, the use of LLM heuristics did not improve the search of the reinforcement learning as seen in Figure 5-2. However, one consideration to take into account is that every time the LLM is asked for guidance it takes 30

Algorithm 1 DDPG [14] with LLM guidance

Randomly initalize critic network $Q(s, a|\theta^Q)$, with weights θ^Q and θ^μ.
Initialize target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
for episode=1 to M **do**
    Initialize random process $\mathcal{N}$
    Receive initial observation state s_1
    for t=1 to T **do**
        if $g \sim \mathcal{U}(0, 1) < \tau$ **then**
            Get action from LLM a_t = LLMGuidance(s_1)
        else
            Select $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        end if
        Execute action a_t, and observe reward r_t, next state s_{t+1}
        Store (s, a, r, s', d) in replay buffer $\mathcal{R}$
        Sample a random minibatch of N transitions (s, a, r, s', d) from $\mathcal{R}$
        Set $y_i = r_i + \lambda Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing loss: $L = \frac{1}{N}\sum(y_i) - Q(s_i, a_i|\theta^Q))^2$
        Update actor policy using the sampled policy gradient:
            $\nabla_{\theta_i^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Update target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
end for

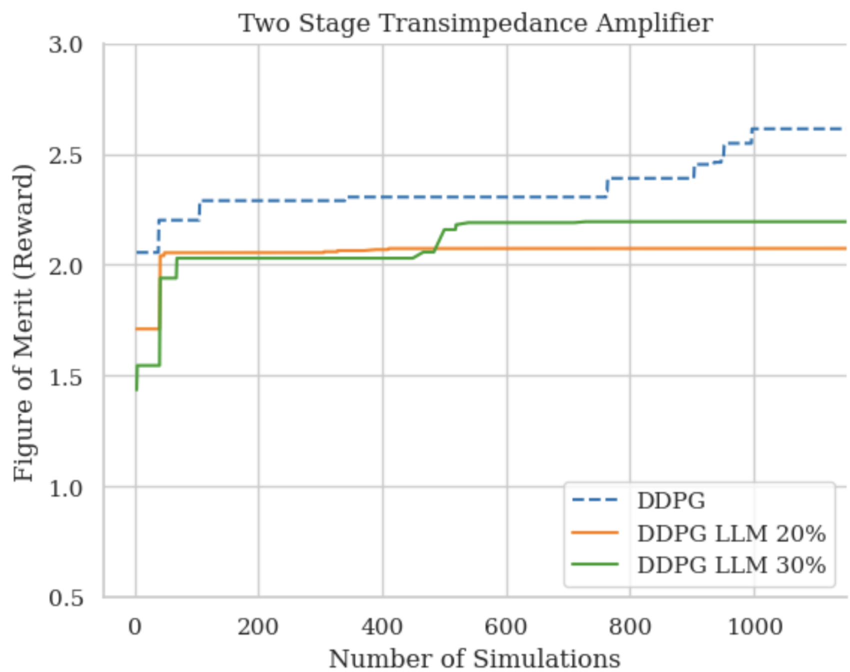


Figure 5-2: LLM Guided DDPG Reinforcement Learning

simulations to return an answer. This is because the LLM's guidance is still given to an evolutionary search optimizer which picks a final set of sizings and rewards to send to the RL training loop. If LLM could pick its own sizings it could reduce the number of simulations between RL policy updates.

Improvements

The way I chose to guide the RL algorithm is only one way to incorporate outside knowledge into an RL training pipeline. Other proposed methods have used LLMs to pick initial higher-level actions and then used an RL agent to search after that subspace [19]. A lot of advancements in RL performance have come from learning from demonstrations. One such proposed method, DQfD leverages human demonstrations to great success [8]. This method could be used for learning how to design circuits if there was a dataset of sizing decisions made by expert designers on different circuits.

Chapter 6

Conclusion

Key Takeaways

The key contributions of this thesis include the evaluation of the analog circuit design capabilities of current state-of-the-art pre-trained large language models. The development of an infrastructure for evaluating analog design tasks using LLMs. The creation of an LLM-empowered optimization Python framework. And finally an implementation of an LLM-RL training procedure for optimizing analog circuits. These efforts lay the groundwork for further advancements in this field Currently, LLMs remain relatively immature in highly specialized domains such as analog design. As the technology evolves, this study can serve as a foundation for achieving more substantial improvements, potentially with the aid of more sophisticated or fine-tuned LLMs.

Future Work

While preliminary experiments using these LLMs showed promising results when it comes to using the models as chip design knowledge bases, further works need to be done to verify these results. Specifically, analyses need to be performed to isolate exactly what kinds of knowledge the models have about the circuits and the design process as well as what parts of the prompt the models are using to generate their response. The limits of the model should also be tested by increasing the complexity of the circuits provided past what was done in this thesis. This will also help determine if the models have a deeper knowledge of circuit design based on whether they are able to correctly analyze uncommon circuits they more likely will not have seen before. So far the experiments have been performed using mostly zero-shot prompting techniques. However, great performance improvements have been seen with Chain-Of-Thought reasoning [28] and other more refined prompting techniques. Furthermore, a lot of LLM improvement is done through fine-tuning [9]. This can improve the performance of the LLM in specialized domains. [12] This strategy could also be leveraged on smaller LLMs allowing the LLM to act more efficiently. The connection between the LLM and the optimization algorithms can also be further improved. The optimization strategies that limit the search space per iteration performed better than a completely random search. However, since current LLMs are not quite adapted to this problem their guidance is not yet better than random. If better guidance could be extracted from future LLMs this approach would be promising. Another initial goal of the thesis was to improve the transparency of circuit optimization through the use of LLMs. However, the explanations provided by LLMs for their decisions could not be trusted to always match their final answers. Nevertheless, this thesis provided a framework for integrating LLM guidance with optimization algorithms. Although the guidance from LLMs may not always be entirely accurate, summarizing or compiling it could still yield valuable insights into the optimization process.

Appendix A

Appendix

A.1 Truncated Netlists Given to LLM

A.1.1 Two Stage Transimpedance Amplifier

Uses open source transistors from EE214 HSpice models, Boris Murmann, January 2010

```
* Two Stage Transimpedance Amplifier


.param w1_2 = %su * width of nmos M1 and M2
.param w3_4 = %su * width of pmos M3 and M4
.param w5 = %su * width of nmos M5
.param w6 = %su * width of nmos M6


.param l1_2 = %su * length of nmos M1 and M2
.param l3_4 = %su * length of pmos M3 and M4
.param l5 = %su * length of nmos M5
.param l6 = %su * length of nmos M6


.param mul2_4 = %s * mutiplies width of nmos M2 and M4

```

```
.param Rf = %s * rf resistor

.param R6 = %s * r6 resistor value


vdd vdd 0 1.8

cdiode in 0 200f

* Stage 1: M1, M2, M3, M4 make a current to voltage
   conversion circuit

* M1 and M2 make one current mirror
M1 in in 0 0 nmos214 w=w1_2 l=l1_2
M2 out1 in 0 0 nmos214 w=mul2_4*w1_2 l=l1_2

* M3 and M4 make the second current mirror
M3 in in vdd vdd pmos214 w=w3_4 l=l3_4
M4 out1 in vdd vdd pmos214 w=mul2_4*w3_4 l=l3_4


* Stage 2:
* M5 makes a common source amplifier
M5 out2 out1 0 0 nmos214 w=w5 l=l5
* M6 is the lead of M5
M6 vdd g6 out2 0 nmos214 w=w6 l=l6

*rf resistor determines the gain
rf x in RF
```

```
r6 vdd g6 R6

cload out2 0 20f

vx out1 x 0

is in 0 ac 1 dc 0 sin (0 150u 1e9)

.end
```

A.1.2 Three Stage Diff Trans Amplifier

Uses open source transistors from EE114 HSpice models, Boris Murmann, October 2008.

```
*** Three Stage Diff Trans Amp ***

.param w1 = %su
.param w2_3 = %su
.param w4_5 = %su
.param w6_7 = %su
.param w8_9 = %su
.param w10_11 = %su
.param w12_13 = %su
.param w14_15 = %su
.param w16_17 = %su
.param l1 = %su
.param l2_3 = %su
.param l4_5 = %su
.param l6_7 = %su
.param l8_9 = %su
```

```
.param l10_11 = %su
.param l12_13 = %su
.param l14_15 = %su
.param l16_17 = %su
.param rb = %sk


*Using resistor (RB) in series with diode-connected MB to set
    VovB

M4 vo1a 0 iina vss nmos114 w='w4_5' l='l4_5'
M6 vo1a vo1a vdd vdd pmos114 w='w6_7' l='l6_7'
M2 iina nbias vss vss nmos114 w='w2_3' l='l2_3'
M5 vo1b 0 iinb vss nmos114 w='w4_5' l='l4_5'
M7 vo1b vo1b vdd vdd pmos114 w='w6_7' l='l6_7'
M3 iinb nbias vss vss nmos114 w='w2_3' l='l2_3'

M10 vo2a vo1a v0d vss nmos114 w='w10_11' l='l10_11'
M12 vo2a vo2a vdd vdd pmos114 w='w12_13' l='l12_13'
M8 v0d nbias vss vss nmos114 w='w8_9' l='l8_9'
M11 vo2b vo1b v0d vss nmos114 w='w10_11' l='l10_11'
M13 vo2b vo2b vdd vdd pmos114 w='w12_13' l='l12_13'
M9 v0d nbias vss vss nmos114 w='w8_9' l='l8_9'

M16 vdd vo2a vouta vss nmos114 w='w16_17' l='l16_17'
M14 vouta nbias vss vss nmos114 w='w14_15' l='l14_15'
M17 vdd vo2b voutb vss nmos114 w='w16_17' l='l16_17'
M15 voutb nbias vss vss nmos114 w='w14_15' l='l14_15'
```

```
*** Bias Circuitry ***
M1 nbias nbias vss vss nmos114 w='w1' l='l1'
rB vdd nbias 'rb'

.end
```

A.1.3 Two-Stage Voltage Amplifier

```
*** Two-Stage Voltage Amplifier ***

* parameters
.param w2_6_8_9_11_12_16_17=%sn * width of nmos M2, M6, M8,
   M9, M11, M12, M16 and M17
.param w1_4_10_15_22=%sn * width of pmos M1, M4, M10, M15,
   and M22
.param w13_14_20_21=%sn * width of pmos M13, 14, 20, and 21
.param w0_3=%sn * width of nmos M0 and M3
.param w5_7=%sn * width of pmos M5 and M7
.param l2_6_8_9_11_12_16_17=%sn * length of nmos M2, M6, M8,
   M9, M11, M12, M16 and M17
.param l1_4_10_15_22=%sn * length of pmos M1, M4, M10, M15,
   and M22
.param l13_14_20_21=%sn * length of of pmos M13, 14, 20, and
   21
.param l0_3=%sn * length of nmos M0 and M3
.param l5_7=%sn * length of pmos M5 and M7
.param mul16_17=%s * mutiplies size of nmos M16 and M17
.param mul12=%s * mutiplies size of nmos M12
.param mul6_8=%s * mutiplies size of nmos M6 and M8
```

```
.param mul2=%s * mutiplies size of nmos M2
.param mul1_4=%s * mutiplies size of pmos M1 and M4
.param mul15_22=%s * mutiplies size of pmos M15 and M22
.param r0_1=%sK * r0 and r1 resistor value
.param c0_1=%sf * c0 and c1 capacitor value



// Cell name: Miller_two_stage
subckt Miller_two_stage vbn vdd vin vip von vop vref
    M17 (net015 net015 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=mul16_17 nf=1
    M16 (net021 net021 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=mul16_17 nf=1
    M12 (net3 vcmfb 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=mul12 nf=1
    M11 (vbn vbn 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=1 nf=1
    M9 (vbp vbn 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=1 nf=1
    M8 (von vbn 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=mul6_8 nf=1
    M6 (vop vbn 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=mul6_8 nf=1
    M3 (net4 vin net3 0) nch l=l0_3 w=w0_3 m=1 nf=1
    M2 (net3 vbn 0 0) nch l=l2_6_8_9_11_12_16_17 w=
        w2_6_8_9_11_12_16_17 m=mul2 nf=1
    M0 (net1 vip net3 0) nch l=l0_3 w=w0_3 m=1 nf=1
    M22 (net35 vbp vdd vdd) pch l=l1_4_10_15_22 w=
        w1_4_10_15_22 m=mul15_22 nf=1
```

```
    M21 (net021 vref net35 vdd) pch l=l13_14_20_21 w=
        w13_14_20_21 m=1 nf=1
    M20 (net015 von net35 vdd) pch l=l13_14_20_21 w=
        w13_14_20_21 m=1 nf=1
    M15 (net18 vbp vdd vdd) pch l=l1_4_10_15_22 w=
        w1_4_10_15_22 m=mul15_22 nf=1
    M14 (net021 vref net18 vdd) pch l=l13_14_20_21 w=
        w13_14_20_21 m=1 nf=1
    M13 (net015 vop net18 vdd) pch l=l13_14_20_21 w=
        w13_14_20_21 m=1 nf=1
    M10 (vbp vbp vdd vdd) pch l=l1_4_10_15_22 w=w1_4_10_15_22
         m=1 nf=1
    M7 (von net4 vdd vdd) pch l=l5_7 w=w5_7 m=1 nf=1
    M5 (vop net1 vdd vdd) pch l=l5_7 w=w5_7 m=1 nf=1
    M4 (net4 vbp vdd vdd) pch l=l1_4_10_15_22 w=w1_4_10_15_22
         m=mul1_4 nf=1
    M1 (net1 vbp vdd vdd) pch l=l1_4_10_15_22 w=w1_4_10_15_22
         m=mul1_4 nf=1
    R0 (net012 net1) resistor r=r0_1
    R1 (net011 net4) resistor r=r0_1
    C0 (net012 vop) capacitor c=c0_1
    C1 (net011 von) capacitor c=c0_1

    IPRB0 (net015 vcmfb) iprobe
ends Miller_two_stage
// End of subcircuit definition.



// Cell name: cmdmprobe
    subckt cmdmprobe in1 in2 out1 out2
```

```
    parameters CMDM=1
        evinj (in2 out2 in1 out1) vcvs gain=CMDM
        vinj (inout in1) iprobe
        vprb (inout out1) iprobe
        fiinj (0 out2) pcccs gain=CMDM probes=[ vprb vinj ]
           coeffs=[ 0 1 1 ]
    ends cmdmprobe
// End of subcircuit definition.

// Cell name: Closed_loop_two_stage_TB
I0 (vbn vdd vir_n vir_p net07 net06 vref) Miller_two_stage
E0 (vip vref net3 vref) vcvs gain=0.5
E1 (vin vref net3 vref) vcvs gain=-0.5
V0 (vref 0) vsource dc=0.9 type=dc
V2 (vdd 0) vsource dc=1.8 type=dc
V1 (net3 vref) vsource mag=1 type=sine
C7 (vir_n vop) capacitor c=1p
C6 (vin vir_n) capacitor c=1p
C4 (vir_p von) capacitor c=1p
C2 (vip vir_p) capacitor c=1p
C0 (von 0) capacitor c=2.5f
C1 (vop 0) capacitor c=2.5f
I1 (vdd vbn) isource dc=10u type=dc
I2 (net07 net06 von vop) cmdmprobe CMDM=-1
R2 (vref vir_n) resistor r=1T
R0 (vref vir_p) resistor r=1T

.end
```

Bibliography

[1] Ashlesha Akella, Abhijit Manatkar, Brij Chavda, and Hima Patel. An automatic prompt generation system for tabular data tasks, 2024.

[2] Mauricio Fadel Argerich, Jonathan Fürst, and Bin Cheng. Tutor4rl: Guiding reinforcement learning with external knowledge. In *AAAI Spring Symposium Combining Machine Learning with Knowledge Engineering*, 2020.

[3] Manish Bhojasia. Analog circuits mcq (multiple choice questions), 2011.

[4] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. Chip-chat: Challenges and opportunities in conversational hardware design, 2023.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[6] Jing Chen, Mohamed Baker Alawieh, Yibo Lin, Maolin Zhang, Jun Zhang, Yufeng Guo, and David Z. Pan. Powernet: Soi lateral power device breakdown prediction with deep neural networks. *IEEE Access*, 8:25372–25382, 2020.

[7] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models, 2023.

[8] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations, 2017.

[9] Hanxu Hu, Pinzhen Chen, and Edoardo M. Ponti. Fine-tuning large language models with sequential instructions, 2024.

[10] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models, 2023.

[11] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. Better zero-shot reasoning with role-play prompting, 2024.

[12] Bhawesh Kumar, Jonathan Amar, Eric Yang, Nan Li, and Yugang Jia. Selective fine-tuning on llm-labeled data may reduce reliance on human annotation: A case study using schedule-of-event table detection, 2024.

[13] Anthony Liang, Ishika Singh, Karl Pertsch, and Jesse Thomason. Transformer adapters for robot learning. In *CoRL 2022 Workshop on Pre-training Robot Learning*, 2022.

[14] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[15] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. Dreampiace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[16] Liane Makatura, Michael Foshey, Bohan Wang, Felix HähnLein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Spielberg, Crystal Elaine Owens, Peter Yichen Chen, Allan Zhao, Amy Zhu, Wil J Norton, Edward Gu, Joshua Jacob, Yifei Li, Adriana Schulz, and Wojciech Matusik. How can large language models help humans in design and manufacturing?, 2023.

[17] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, page 50–56, New York, NY, USA, 2016. Association for Computing Machinery.

[18] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. Chip placement with deep reinforcement learning, 2020.

[19] Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling, 2023.

[20] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman,

Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Wein-

mann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[21] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. Language models as knowledge bases?, 2019.

[22] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[23] Lorenzo Servadei, Edoardo Mosca, Michael Werner, Volkan Esen, Robert Wille, and Wolfgang Ecker. Combining evolutionary algorithms and deep learning for hardware/software interface optimization. In *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2019.

[24] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[25] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning, 2020.

[26] Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. Learning to design circuits, 2020.

[27] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

[28] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[29] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2024.

[30] Wei Ye, Mohamed Baker Alawieh, Yibo Lin, and David Z. Pan. Lithogan: End-to-end lithography modeling with generative adversarial networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[31] Guo Zhang, Hao He, and Dina Katabi. Circuit-GNN: Graph neural networks for distributed circuit design. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7364–7373. PMLR, 09–15 Jun 2019.

[32] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey, 2023.