Title
Pre-training Agents for Design Optimization and Control

Author
Hakhamaneshi, Kourosh

Publication Date
2022

Peer reviewed|Thesis/dissertation

Pre-training Agents for Design Optimization and Control

by

Kourosh Hakhamaneshi

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering & Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Vladimir Stojanović, Co-chair
Professor Pieter Abbeel, Co-chair
Professor Xin Guo

Spring 2022

Pre-training Agents for Design Optimization and Control

Abstract

Pre-training Agents for Design Optimization and Control

by

Kourosh Hakhamaneshi

Doctor of Philosophy in Engineering - Electrical Engineering & Computer Sciences

University of California, Berkeley

Professor Vladimir Stojanović, Co-chair

Professor Pieter Abbeel, Co-chair

In recent years, we have seen tremendous benefits from pre-training neural networks to learn representations that are transferable to unseen downstream tasks in both vision and NLP. However, this paradigm of learning has not been much explored for decision making such as design optimization or control. In this thesis, we outline two problem settings that could benefit from pre-training in the context of decision making. First, we describe a setting for automated design optimization, in particular circuit design optimization, where prior domain-specific data can be used to effectively improve the sample efficiency of model-based optimization methods. This thesis presents novel ideas along with empirical and theoretical analysis on how to boost sample efficiency of model-based evolutionary algorithms as well as Bayesian optimization methods. In the second problem setting, we will discuss how we can leverage unsupervised pre-training from large task-agnostic datasets to extract behavioral representations and do few-shot imitation learning. We find that pre-training agents to extract skills is a practical direction for preparing them for few-shot imitation when example demonstrations from the new task are scarce.

To my amazing wife, Yasi

Contents

List of Figures

List of Tables

Acknowledgments

I want to dedicate a few paragraphs of this dissertation to thank the folks without whom this Ph.D. would not have been possible. First and foremost, I want to thank my advisors Vladimir Stojanović and Pieter Abbeel. Working with two professors from entirely two different disciplines was one of the most challenging but at the same time most rewarding aspects of my doctoral studies. Vladimir! Thank you for giving me the freedom in my research and for helping me expand my skills beyond circuit design. Your research group is one of the most diverse and educative groups in BWRC and I enjoyed being part of it. Pieter! Thanks for allowing me to join your group in the middle of my Ph.D. and for trusting me with my research agenda. I enjoyed being part of your research group and learned so many things that make me regret why I did not take the initiative of joining earlier. I'm grateful for your initiatives in creating a more collaborative and productive research group as a whole that not only produces top-notch research but also gives value to each individual's social life and well-being, especially in the midst of the pandemic crisis. I also want to thank you for helping me out through the job search and giving me several amazing pointers to new connections whom I can expand my career with.

During my Ph.D., I had several mentors who have helped me immensely in navigating my graduate studies. On the BWRC side, Sajjad was my research buddy when I joined Berkeley and passed on all the experiences he had from his Ph.D. So thank you for that. I want to also thank those who helped me ramp up on Berkeley Analog Generator (BAG). Eric Chang, Sidney Buchbinder, Krishna Settaluri, and Pavan Bhargava! Thank you for helping me learn BAG and contribute to it. Ranko! Thanks for seeding the idea of using reinforcement learning in circuit design and helping me brainstorm ideas in the first few months. From the BAIR side, I want to thank Aditya Grover, who has helped me a lot on the JUMBO project and thought me how to write good papers. I'm grateful for the many hours that we spent together reviewing the paper and its rebuttals. Though it did not get into the venues we had hoped for, it certainly had a lot of insightful takeaways for me, at least. I want to also thank Misha Laskin for teaching me how to expand a research idea from the initial seed to a well-executed paper. Interacting with you has been certainly one of the most rewarding aspects of my time at Berkeley, and last but not least Mariano Phillip! Thanks for allowing me to join intel AI labs for the summer internship and to pursue my research agenda. I have also had the pleasure of working with a lot of good collaborators in my projects: Keertana Settaluri, Nick Werblun, Albert Zhan, Ruihan Zhao, Phillip Wu, Marcel Nassar, Catherine Cang, Aravind Rajeswaran, and Igor Mordatch. Thank you all.

Mom and Dad! Thank you for encouraging me to pursue my dreams even if it meant being far from your loved ones. I haven't enjoyed your company for so long and I miss that but hopefully, this year (2022) things are different and you can visit us in the US. My in-laws, Sima and Farzad! Thanks for your constant support and always offering to help with getting something off our shoulders. And finally to my amazing wife, Yasi! I am forever in your debt for your constant support and the sacrifices you have made for us to get to where we are now and where we'll be in the future. I am forever grateful for your decision on joining

me on this journey of 5 more years of school and coming to the funky city of Berkeley. You kept me sane through all these years and brought joy to my life, especially throughout the pandemic, and I appreciate that. I very much look forward to what lies ahead of us and I cannot say I'm not excited about that.

Chapter 1

Introduction

The de-facto standard approach of Deep Learning in recent years to solve tasks is to collect a large amount of labeled data for that task and train a deep neural network end-to-end. This paradigm shift in problem-solving has completely changed the way humans think about problems nowadays. In the "Good" old days, machine learning practitioners had to carefully investigate the data and hard-code their domain knowledge into the program they wrote for solving particular tasks; however after the ImageNet moment in 2012 [65] they have realized that a combination of large labeled datasets, deep neural networks, and access to large amounts of computational resources can be the new recipe for solving problems in different applications, ranging scene understanding [20, 112], playing video games [125], and object grasping with robotic arms from visual inputs [57].

While deep neural networks can be trained from scratch with minimal human engineering on priors, this *tabula rasa* approach makes them very data-hungry in practice. The *tabula rasa* training approach starts to fall short when there are no large datasets available for the problem at hand.

However, from the perspective of natural intelligence, humans do not seem to operate in a *tabula rasa* training regime. For one thing, evolution has prepared human brain and intelligence system to be able to quickly learn and adapt to new scenarios that are not seen before, without needing large amounts of labeled data. In addition to this preparation, for a while after birth, humans go through an unsupervised learning stage where they learn only from observing the world around them and how the entities in the world interact with one another. Only after these many preparations is when they get supervised training via their parents or their schools.

Researchers in the field of artificial intelligence have also embraced this idea of pre-training over the years. The most obvious manifestation of this idea has been in computer vision where a large deep neural network is pre-trained on a large labeled dataset like ImageNet to predict the output classes. Then, the pre-trained backbone of the neural network is fine-tuned on new visual prediction tasks such as medical imaging [100] which is entirely different from the distribution of natural images seen in pre-training. Despite this disparity, the method has shown very promising results in the ability of neural networks to extract

general visual representations. More recently, we have seen a surge of methods in using self-supervised pre-training for both vision and language where a labeled dataset is not even needed. Most prominent works in this direction are MOCO [47] in vision [18] and BERT [18] and GPT-X [98, 10] models in NLP.

In this thesis, we consider two problems that have not seen much benefit from the pre-training paradigm over the years, but can certainly benefit from better sample efficiency via pre-training. The first problem that we will consider is design optimization, especially analog and mixed-signal circuit design optimization, which is a good fit for applying modern machine learning methods. The challenge in applying common deep learning practices is that the data collection in this domain is quite expensive and requires an efficient learning system to automate the design process. The second problem is using pre-training in control and reinforcement learning where the goal is to prepare a multi-task agent for few-shot adaption via imitation on new tasks that are not seen during pre-training.

Thesis Contributions and Outline

In **Chapter 2**, we introduce the field of analog circuit design automation and cover the prior works that have been done according to the literature.

In **Chapter 3**, we explain what software tools we have created to facilitate the experimentation and data collection for analog circuit design automation using deep learning. We present the basis of a framework that is built on Berkeley Analog Generator [12] which abstracts away the parametrization and execution of layout generation and metric measurement in analog-mixed signal circuits on real process development kits (PDKs). This framework lets us formulate layout optimization as a black-box optimization in Python. We also present an open-source counterpart built on `NGSPICE` which allows us to quickly iterate on algorithmic ideas and development of robust optimization algorithms without having to worry about issues with licensing or slowness of simulators.

In **Chapter 4**, we propose a novel modeling approach to boost the sample efficiency of existing evolutionary algorithms. Our method replaces the simulator-based discriminator in the selection mechanism of the evolutionary algorithms with a learned deep neural network that can accelerate function evaluation. We show the benefits of our method by applying the algorithm to real-world complicated circuit design problems which also considers the parasitic effects introduced by layout. Using this method we can optimize a photonic link receiver front-end layout, based on high-level specifications in almost a day of compute (27 hours). This work was published in ICCAD 2020 [38].

In **Chapter 5**, we explain how we can pre-train graph neural networks to learn better models that further boost the sample efficiency of evolutionary algorithms. We found that prediction of the DC simulation results of an analog circuit can serve as a scalable pre-training objective that lets us learn re-usable circuit embeddings across many circuits. This pre-training allows our downstream models to be less data-hungry and thereby improve the sample efficiency of our model-based optimization algorithms [41].

In **Chapter 6**, we investigate Bayesian optimization as another optimization paradigm that uses probabilistic models to provide a very sample efficient optimization framework for problems where the evaluation of the function values is expensive. We propose JUMBO, a multi-task Bayesian optimization algorithm that can efficiently re-use the knowledge in additional offline data based on a combination of acquisition signals derived from training two Gaussian Processes (GP): a *cold-GP* operating directly in the input domain and a *warm-GP* that operates in the feature space of a deep neural network pre-trained using the offline data. Such a decomposition can dynamically control the reliability of information derived from the online and offline data and the use of pre-trained neural networks permits scalability to large offline datasets. We show how such the pre-training mechanism can improve the sample efficiency of existing Bayesian optimization methods on a circuit design problem as well as a hyperparameter optimization problem. This work was presented in the AAAI 2022 conference, the workshop track on AI for decision optimization (AI4DO) [40].

In **Chapter 7**, we explore the idea of skill extraction pre-training for few-shot imitation learning. We propose a new method (FIST), an algorithm that extracts skills from offline data and utilizes them to generalize to unseen tasks given a few downstream demonstrations. FIST learns an inverse skill dynamics model, a distance function, and utilizes a semi-parametric approach for imitation. We show that FIST is capable of generalizing to new tasks and substantially outperforms prior baselines in navigation experiments requiring traversing unseen parts of a large maze and 7-DoF robotic arm experiments requiring manipulating previously unseen objects in a kitchen. This work was published in the International Conference on Learning Representations (ICLR) 2022 [39].

Finally, in **Chapter 8**, we conclude with future directions for how large scale pre-training can be used for both design optimization and control.

Chapter 2

Integrated Analog Mixed Signal Circuit Design Automation

This chapter presents the motivation behind the research on analog-mixed signal (AMS) circuit design automation. It outlines the market and technological evolution, characterizes the analog IC design, and finally discusses the available solutions.

2.1 Market Overview

The Microelectronic industry has significantly evolved over the past years, driven by an ever-increasing demand for machine learning (ML) and internet of things (IoT) applications, requiring many functionalities to be integrated into a single System-on-Chip (SoC) solution. SoCs of today's generation integrate a massive number of digital computational cores (i.e. CPU, GPU, and neural processing units) with an increasing number of analog-mixed signal and radio frequency (RF) components (i.e. radio transceivers, sensors, power regulators, high-speed IOs, etc.) on the same chip. This makes the design and verification process exponentially more convoluted than before and also prone to more errors. Figure 2.1 shows the growth of the cost of building a single system-on-chip. As we can see, the majority of the cost is dominated by design, verification, and software engineering efforts required for building the chip. Shortening this design cycle can certainly improve chip development margins, increasing the profits of companies that specialize in the design of these system-on-chips.

Although the core functionality is typically driven by the digital cores, the critical path for errors and performance tends to be in the analog part. This is due in part to the differences in the maturity levels of CAD tools in digital and analog design. Well-established practices supported by well-defined automated synthesis methodologies and tools exist for digital design. However, analog design, due to the higher sensitivity of the design to parameters and the fabrication process, requires extensive, skilled manual labor. It is not as modular as digital design, and as a result, a standard cell library of sub-blocks cannot be used.

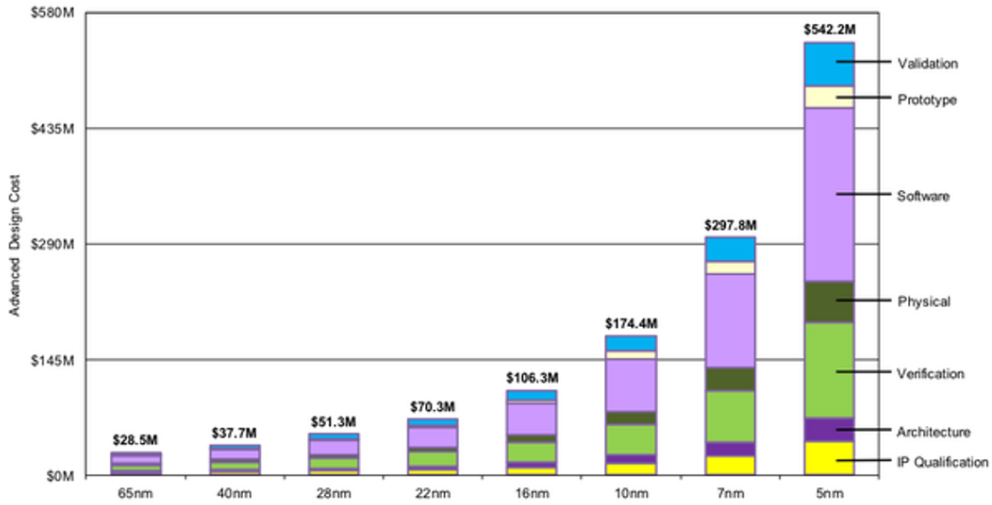


Figure 2.1: The exponential growth of cost of developing system on chips. It costs about half a billion dollars to build a large system on chip on today's 5nm technology most of which is dominated by design and verification cost.

2.2 Analog IC Design Flow

As was mentioned in the previous section, AMS design requires more manual labor than digital design. In the digital world, signals are more tolerant to noise and the information is carried around in discrete levels of voltage or current with a large tolerance range. The behavior of such circuits is defined by Boolean algebra, and the timing information is also modular and defined for a library of standard cells. The optimization problems are well-defined and therefore automation tools are much easier to be developed.

In AMS circuits, however, information is conveyed through a continuum of (typically small) values, and therefore second and third-order effects which were ignored in digital design become a problem in AMS design; issues like, non-linearity, devices going into their non-linear region, matching between devices, effects of noise, layout wiring parasitics and loading, etc. have to be considered. Figure 2.2 [5] depicts a clear picture of the design effort per unit area between analog to digital circuits in a typical commercialized IC. Despite consuming a smaller area, the design and validation effort for AMS circuits, due to the impact of second and third-order effects, is much larger than its digital counterparts.

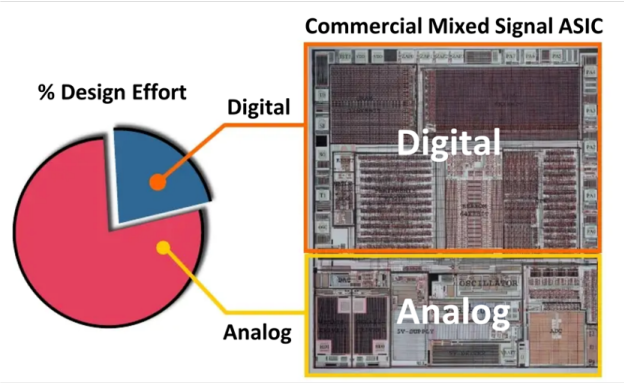


Figure 2.2: The design effort per unit area between digital and analog circuits. Despite consuming a smaller area the design effort for analog circuits, due to the impact of second and third order effects, is much larger than digital counter-parts [5]

To cope with these issues designers have to adopt top-down design and bottom-up verification methodologies. This means that they start the design flow with top-level behavioral modeling and hierarchically move down and verify the performance and behavior at each step with different levels of abstraction. To do so, they start with system-level abstraction, verification, and architecture exploration until they settle on the AMS sub-blocks and their interaction with each other. The agile IC development flow requires designers to then use dummy or semi-realistic sub-blocks as the placeholder for the circuits that will be eventually designed later and verify the entire system-level flow with the behavioral models that they expect from each sub-block. This way they can catch system-level problems early on as the sub-blocks evolve over the design procedure. This process is then repeated for lower-level sub-blocks (which have their own sub-blocks) until they reach the actual device and layout level design.

As designers move down the hierarchy, they have to re-verify their assumptions on the higher levels and if something does not behave properly re-design the lower levels. The number of iterations in verification and design tends to increase as designers get to the layout. This is largely in part due to the domination of layout parasitics in the determination of circuit performance. Therefore, designing the entire system involves many iterations with human experts exploring the complex multi-dimensional design space. Besides the large number of iterations, simulation will also get slow as designers move down from behavioral simulations to the device and layout levels.

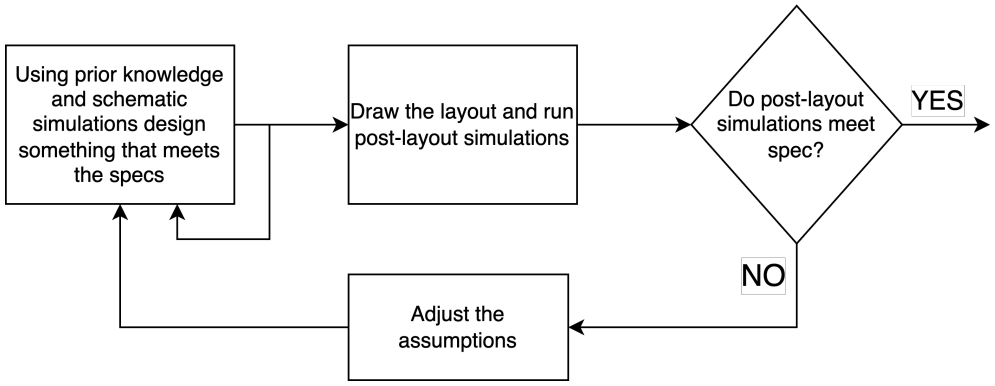


Figure 2.3: An abstraction of the flow for design and verification of analog-mixed signal circuits. After verifying the behavioral and functional performance, the layout is drawn and if the post-layout simulations do not satisfy the specs the assumptions are adjusted and the design is iterated all the way from the top.

2.3 History of Analog IC Design Automation

Over the ages of the semiconductor industry, researchers have tried an assortment of approaches to automate the AMS design in different stages. However, CAD tools have not been following the semiconductor technology at the same pace, resulting in increasing the price of developing state-of-the-art SoCs. In this section, we will briefly discuss available CAD tools and some automation procedures proposed over the years.

CAD Tools for Analog-Mixed Signal Design

An improved but yet limited degree of automation is supplied by the use of a CAD methodology that involves the integration of one or more mature CAD tools into a flow. One of the most known CAD tools is the Cadence® Virtuoso platform which is composed of a set of integrated circuit tools that cover all the stages, from the schematic to the layout. Apart from the Composer schematic editor, Cadence Virtuoso includes a high accuracy circuit simulator, like Virtuoso Spectre which is usually used at transistor-level simulation, a layout editor, and layout verification tools that implement the three different phases of the layout process: the design rule checking (DRC), layout versus schematic (LVS) and parasitic extraction (RCX). Additionally, the system-level analog behavioral descriptions may be simulated with the System-Verilog simulator. These design management platforms are a valuable help in analog integrated circuit design but they are still far behind the development stage of design automation tools already available for digital design [5].

Design Automation Methodologies

Automating AMS IC design is not a novel idea by itself and has been around for over twenty years. An excellent survey about these efforts has been done in [106]. We can divide the previous approaches into the following subcategories:

Topology Selection Approaches

In this flavor of approaches, the goal is to infer the topology based on some criteria either from a library or by composing sub-components together and forming more complex ones. To name a few of these efforts, in IDAC [17] the decision is taken directly by the designer. Heuristic rules have been used in the first attempts by [43, 63], to automate the topology selection task. The tool [122] uses fuzzy logic reasoning to select one topology among a fixed set of alternatives. The decision rules are introduced by an expert designer or automatically generated through a learning process. Another method comprises computing the feasible performance space for each topology within the library and then comparing it with the desired performance specs [32]. A different method consists of combining the topology selection with the device sizing task and employing an optimization-based approach by [66] using genetic algorithms.

As was mentioned earlier, these approaches have several problems: The new generation of designs is far more complicated to be supported by a predefined set of libraries. Most of these approaches demonstrated their usefulness in the design of operational amplifiers which is not as complicated as today's practical circuits. On top of that, changing the technology requires new rounds of library generation, which might be time-consuming. Additionally, all of these methods do not consider layout effects which drastically change the performance of the circuits.

Circuit Sizing/Optimization Approaches

These approaches typically cast analog sizing into an optimization problem and seek to find the parameters that satisfy their objective. In many cases, the optimization formulations do not consider all practical and non-ideal phenomena in circuits, leaving the burden on analog designers to set the constraints properly, and guide the optimization procedure. Therefore, it is easier for analog designers to just directly solve the optimization themselves rather than codifying all overt design criteria (i.e. matching of layout, transistors not being in the linear region, etc.), as well as performance metrics (i.e. gain, bandwidth, etc.).

A lot of these readily assumed design metrics are often hard to codify, either because they cannot be captured by plain simulations (i.e. matching) or they need a specialized expensive simulation (i.e. Monte Carlo simulations for matching). Some of these "hard-to-codify" metrics come from layout methodologies that designers use because they are "safe" and proven to be functional (i.e. how to use common centroid layout procedure for current mirrors to ensure matching). Besides these, when utilizing such tools, designers need to get used to fully specifying the constraints of the problem to prevent the tool from exploiting

the unspecified constraints, which takes a long time since the standard design practice is to iteratively solve for most important constraints first and then fine-tune the design on the second and third-order constraints. [5],[106] divides the circuit sizing approaches into the following categories:

Equation-Based Methods. These approaches model the circuit behavior as analytical formulations and the objective is to solve an equation-based optimization problem using numerical methods.

Some of the most relevant approaches are [63, 44, 82, 33, 49]. This approach has the advantage of reduced evaluation time. The main drawback is that analytical models have to be used to derive the design equations for each new topology and despite recent advances in symbolic circuit analysis [128, 128], not all design characteristics can be easily captured by analytic equations. The approximations introduced in the analytic equations yield low accuracy models for the design, especially in complex circuits. These methods are obsolete and cannot guarantee the accuracy of the solution.

Simulation-Based Methods. Black-box optimization methods like [81, 122, 92, 64, 66] consist of using a simulation core in the inner loop for circuit evaluation. This gives us a generic framework, independent of circuit topology (compared to equation-based methods), for high accuracy evaluation of circuits.

Despite the benefits, these methods are computationally expensive and time-consuming to run, especially for larger, more complex circuits. Nevertheless, a key difficulty is that the AMS design problem, with all the involved design knowledge and heuristics, has to be formulated as an optimization problem, which often presents a high barrier to entry for using a circuit-sizing tool. Therefore, sample efficiency and short convergence time are determining factors for these approaches. On the other hand, all the previous work in this area has been targeted for schematic-level simulations, and layout parasitic effects have been considered only outside of the optimization loop. In modern technologies, the layout effects create a significantly large difference from the schematic-level design that they need to be included in the design optimization loop.

Machine Learning-Based Methods.

To improve the convergence of these methods, many of the previous works such as [3, 2, 133] have tried to model the behavior of the circuit to be optimized, by a machine learning mechanism to quickly evaluate the performance of a specific set of design parameters, essentially replacing the simulator's long evaluation cycles.

In [2] a learning tool based on support vectors machines (SVM) is used to represent the performance space of analog circuits. Based on the knowledge acquired from a training set, the performance space is modeled as mathematical relations translating to analog functionality. SVMs are trained with simulation data, and false positives are controlled based on a randomized testing procedure.

[133] presents a performance macro-model for use in the synthesis of analog circuits based on a neural network approach. The basis of this mathematical model is a neural network model that, once constructed, may be used as a substitute for full SPICE simulation, to obtain an efficient computation of the performance estimator. The training and validation

data set is constructed with discrete points sampling over the design space. The work explores several sampling methodologies to adaptively improve model quality and applies a sizing rule methodology to reduce the design space and ensure the correct operation of analog circuits.

The issue with almost all of these methods is that they require a lot of sampling points in their training data to build a model that is accurate enough to replace the simulator. Nevertheless, good design points tend to be a small subset of the extremely large multi-dimensional space. Hence, despite the high overall accuracy of the model, the precision in the regions of good designs can be low, resulting in inaccurate models in regions that designers care about. Also, similar to previous methods, these approaches have mainly modeled the schematic behavior of circuits. Training with layout effects can become infeasible using these approaches since a large dataset needed for training cannot be obtained because of the long simulation time of each circuit instance.

Generator-Based Approaches

In this approach, instead of optimizing circuit sizing objectives, the designer codifies the step-by-step sizing and design process they pursue in a parametrized format called *generators*. In other words, the input to a *layout generator* script is the parameters of devices (i.e. number of unit widths, number of fingers, etc.) and the output is the associated layout in a particular technology. Following this strategy, we can capture all layout constraints (i.e. matching, sharing drain/sources to reduce parasitics, etc.) in a programmable, reusable fashion and port a layout to new technologies with just a push of a button.

Berkeley Analog Generator (BAG) [12] is a framework for the development of process-portable AMS circuit generators. It has multiple layout engines that provide designers with the necessary application programming interface (API) to capture their layout procedure for analog as well as custom digital circuits. It also provides a coherent environment for design verification in a Python-based language. Designers can develop *schematic generators* which essentially generate the schematic of a design in a parameterized way. They can then run layout-vs-schematic (LVS) checks and design-rule checking (DRC) directly from the same Python environment, and also run testbenches associated with verification of the blocks. They can also leverage objected-oriented programming (OOP) features of Python to extend and reuse other designers' codes. In summary, it provides a convenient interface for designers to specify their layout, design, and verification methodologies.

Chapter 3

Developing Software Tools for Circuit Design Automation

This chapter presents the necessary tools that were developed to facilitate the research progress made by using AI for analog circuit design. We will present the following tools and how we integrated them into a machine learning framework for black-box optimization formulation of analog sizing:

- Black-box Evaluation Engine
- Berkeley Analog Generator (BAG)
- NGSpice simulator

3.1 Motivation

The progress made in deep learning in recent years is largely due to access to large curated datasets that have become the standard for testing the capability of Neural networks in predictions (e.g. ImageNet [65]). Also in control and reinforcement learning, simulator environments such as OpenAI gym [9], DM Control [121], and more recently ISAAC GYM [75] along with many other simulator environments have made significant contributions to the progress in the field.

The analog circuit design automation community however has not seen such benefits at large from modernized tools that can be used as the backbone of developing new machine learning systems to accelerate the design workflows. Part of this thesis's contribution is to develop and integrate existing simulation pipelines into a software interface that is amenable to machine learning formulation and standardizing simulation tools used for machine learning-based analog circuit optimization problems.

3.2 Definition of Analog Sizing Problem

Before explaining how the Black-box evaluation engine is designed we should understand the problem formulation. The objective in analog circuit design is usually to minimize one figure of merit (FOM) subject to some hard constraints (strict inequalities). For instance, in op-amp design, the objective can be minimizing power subject to gain and bandwidth constraints. However, in practice, there is also a budget for metrics in the FOM (i.e power less than 1mW). Therefore, the optimization can be rephrased as a constraint satisfaction problem (CSP) where the variables are the circuit's geometric parameters, and outputs are specifications of the circuit topology. Designers can always tighten the budgets to see if there is any other answer with a better FOM that meets their needs.

However, in cases where there is no feasible solution to a CSP, designers still prefer to know which solutions are nearly satisfactory to gain insight into which constraint can be adjusted to satisfy their needs.

With this in mind, we define the following non-negative cost function where finding the zeros is equivalent to finding answers to the CSP problem. If no answer exists, the minimum of this cost and the non-zero terms can give insight about which metrics are the limiting factors:

$$cost(x) = \sum_i w_i p_i(x) \tag{3.1}$$

where x represents the geometric parameters in the circuit topology and $p_i(x) = \frac{|c_i - c_i^*|}{c_i + c_i^*}$ (normalized specification error) for designs that do not satisfy constraint c_i^*, and zero if they do. c_i denotes the value of constraint i at input x and is evaluated using a simulation framework. c_i^* denotes the optimal value. Intuitively this cost function is only accounting for the normalized error from the unsatisfied constraints, and w_i is the tuning factor, determined by the designer, which controls prioritizing one metric over another if the design is infeasible. With this problem definition in mind, we can now understand the design of the Black-box Evaluation Engine.

3.3 Black-box Evaluation Engine

Our software design philosophy for circuit design optimization is simple. We isolate the development of the optimization algorithm and the simulator fidelity that is used for candidate evaluation by creating an interface for the black-box evaluation. This way we can make an abstraction for the black-box evaluator and create different instantiations with *any* simulator engine quickly without the need to modify the algorithm. We can even wrap the black-box evaluator engine into an OpenAI gym wrapper for using reinforcement learning (RL) on the problem [109].

Figure 3.1 shows a diagram of this design philosophy. Since the algorithm only interacts with certain general attributes of the black-box engine it never needs to be aware of the underlying details of the simulator engine.

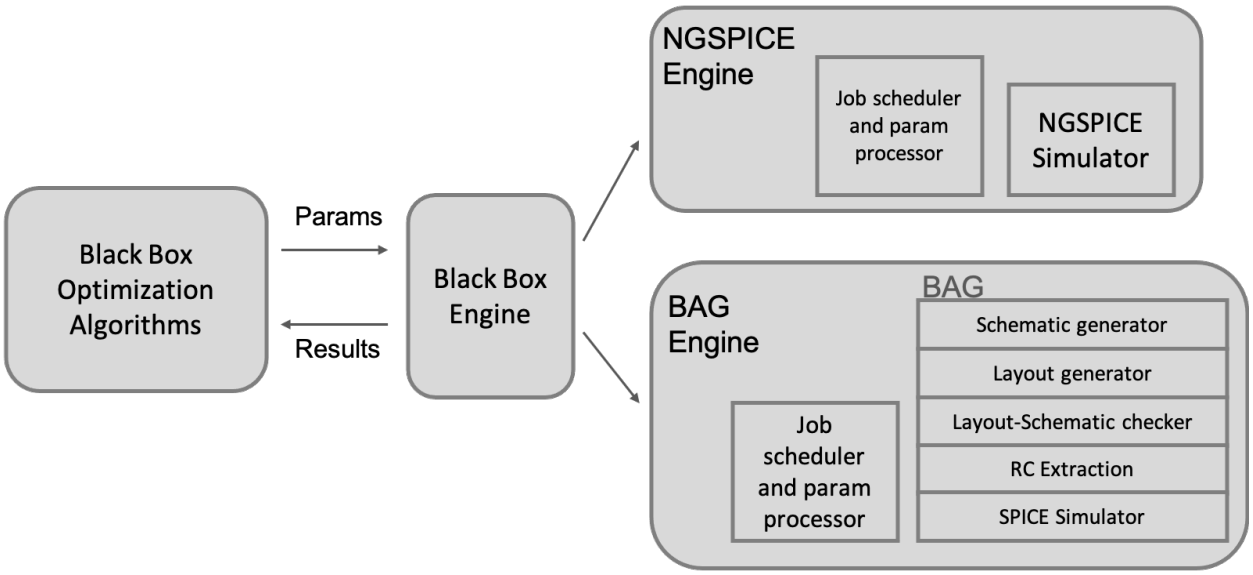


Figure 3.1: Separation of optimization algorithms and black-box evaluations engine

Black-box Evaluation Engine[1] module is responsible for running and querying the evaluation core. It also handles the parallel evaluation of several candidates using Python's multiprocessing and multithreading API [2]. This module is initialized by a specification file that has all the information about the parameters to optimize, their range, the circuit specification, and how to evaluate the cost function. The interface functions are the following:

```
def generate_rand_designs(n, evaluate) -> Sequence[Design]:
    """
    Generates a random sequence of Design samples.

    Parameters
    ----------
    n: int
        the number of individuals in the database.
    evaluate: bool
        True to evaluate the value of each design and
↪   populate its attributes.
    seed: Optional[int]
```

[1]For references look at https://github.com/kouroshHakha/blackbox_eval_engine
[2]https://realpython.com/python-concurrency/

```
        The initial seed for the random number generator.

    Returns
    -------
    database: Sequence[Design]
        a sequence of design objects
    """

def evaluate(self, designs: Sequence[Design]) ->
↪   Sequence[Design]:
    """
    Evaluates (runs simulations) a sequence of design
↪   objects, while resetting the state of designs.
    """

def compute_penalty(self, vals: SpecSeqType, key: str) ->
↪   SpecSeqType:
    """
    Implement this method to compute the penalty(s) of a
↪   given spec keyword based on the provided numbers for that
↪   specification.
    Parameters
    ----------
    vals: SpecSeqType
        Either a single number or a sequence of numbers for a
↪   given specification.
    key: str
        The keyword of the specification of interest.

    Returns
    -------
        A single number or a sequence of numbers representing
↪   the penalty number for
```

```
        that specification
    """
```

We define the black-box's search space with a discrete grid corresponding to each degree of freedom we have for the problem. For example, the following snippet of code shows part of the configuration file that describes the search space of an amplifier. `mp1: [1, 100, 1]` essentially means that `mp1` is a parameter in design that can vary from 1 to 100 with steps of 1. This input definition can only present the integer-valued variables, however, it is in line with the modern analog design practices. Typically, in modern technologies that are FinFET-based, only a discrete set of values are allowed.

```
    mp1:  !!python/tuple [1, 100, 1]
    cc:   !!python/tuple [0.1e-12, 10.0e-12, 0.1e-12]
```

We also define the required specification of the circuit with a hash map to inform the black-box of the metrics that it should keep track of from the simulator's output, and what the requirement for each specification is. The following snippet of code shows an example configuration for an amplifier: `pm: [60.0, null, 1]` means phase margin (pm) should be **at least** 60°, `tset: [null, 60.0e-9, 1]` means settling time (tset) should be **at most** $60ns$. The third element in the specification tuple determines the weight of that specification (w_i) in the computation of the penalty function according to the formulation in Equation 3.1.

```
pm: [60.0, null, 1]
tset: [null, 60.0e-9, 1] #Seconds
```

At the core of this API, we represent each candidate with an object from `Design` class. Each design object is a list of indices that correspond to the index within the grid vector of the search space for each variable. Design objects allow the user to seamlessly switch between different representations, from the indices to values, to specifications, and so on.

```
class Design:
    ...
    @property
    def value(self):
        """ Returns the index representation of the Design
        ↪  object. """
```

```
    @property
    def value_dict(self):
        """ Returns the value representation of the Design
        ↪ object. """

    @property
    def specs(self):
        """ Returns the achieved specifications of the Design
        ↪ object. """
    ...
```

There are two different evaluation cores implemented in this framework: The first one is an `NGSPICE` evaluation engine. `NGSPICE` is an open-source SPICE-like simulator that can be used on any machine without license requirements. It is fast since it does only schematic simulation and reading the data is easy. The second one is `BAG` integrated evaluation engine which can generate the layout, schematic, run LVS and RC extraction, and finally run simulations with post extracted results. This evaluation engine is expensive in run-time and is tied to a particular foundry process design kit (PDK) which is less portable due to licensing and intellectual property requirements.

3.4 Schematic Simulator engine using NGSPICE

To be able to quickly iterate on the optimization algorithms development cycles and not be limited by environment implementation details such as license requirements for doing layout simulation or the slowness of post-layout simulation run-times, we have developed an evaluation engine based on an open-source fast simulator called `NGSPICE` that can do only schematic simulations on open source predictive technologies (BSIM). We used this simulation framework for open-sourcing our code-bases with runnable examples without worrying about the proprietary information of PDKs.

Figure 3.2 shows how the `NGSPICE` evaluation engine is setup. With this framework, we first create a template netlist description of the circuit that we want to optimize, and then the API will use that template to convert the input parameters of a given design to the correct netlist that can be run by the NGSPICE simulator. Then the user-defined post-processing functions are used to parse the output results of the simulation and convert them to the final design metrics for each design. The computed output metrics are then passed back to the algorithm to make new candidate selections for evaluation.

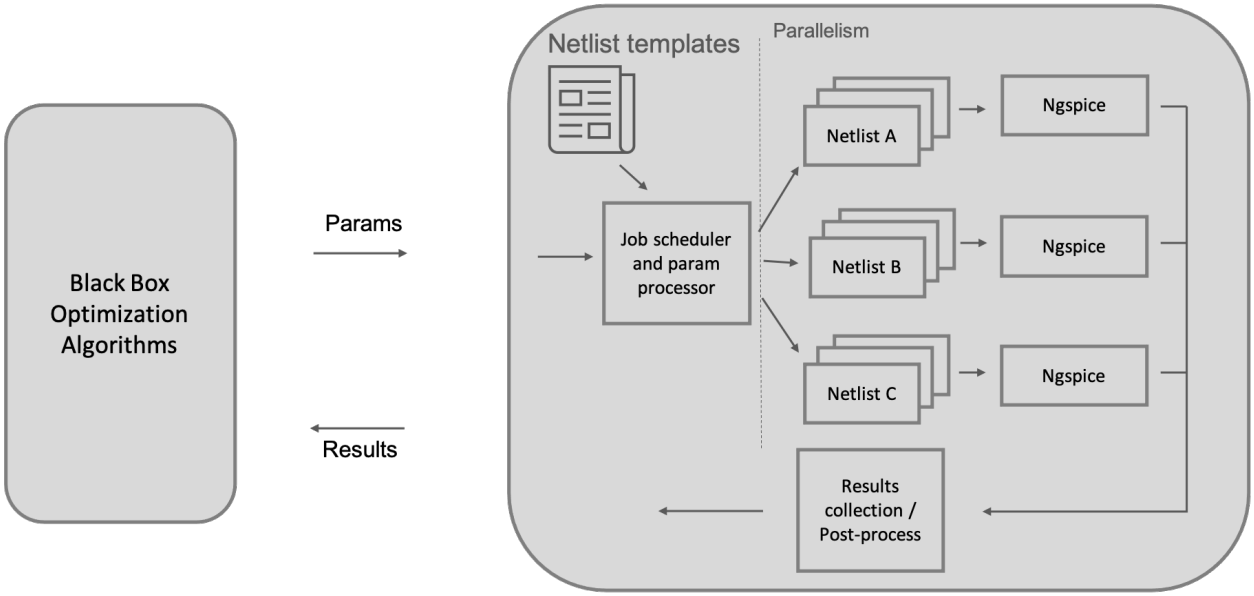


Figure 3.2: The `NGSPICE` flow diagram.

3.5 Layout Simulator engine Using BAG

The difficulty in analog circuit design is rooted in the layout automation part. Ultimately, our method should be able to consider layout effects in the design loop. Therefore, we create an evaluator engine that directly optimizes the parameters that are fed in as the configurations to a BAG generator that generates the layout, schematic, and all the collateral for design verification including post-layout simulation results.

To this end, we used a BAG API called `DesignManager` (available only in BAG2.0). This class manages and orchestrates the design generation jobs. In this class, we have to essentially define how we should convert the input parameters to the layout, schematic, and circuit test benches for measuring the specifications. For complete implementation details, we refer the reader to the code-base at `bb_eval_engine/circuits/bag/DeepCKTDesignManager.py`

We evaluate the layout in two phases: In the first phase, we complete the generation of the layout and then sequentially run LVS and RC extraction. If any of these steps fail, we will not proceed to the next phase. Phase 2 is when we create the test benches and run the simulations with the post-extracted collateral from Phase 1. During parallel execution, we aggregate the results of Phase 1 and Phase 2 in a single list, in the same order that designs were ordered. If phase 1 failed the corresponding entry will contain a `Phase1Error` exception. This way we can filter out designs that failed during simulation from those that succeeded without disrupting the job flows (See `bb_eval_engine/circuits/bag/bagEvalEngine.py` for more details). Figure 3.3 shows an example of the job flow for simulating using layout. Moving forward, we recommend the users to use BAG3.0 instead of BAG2.0 since it has a

better API for handling design simulation and job orchestration.

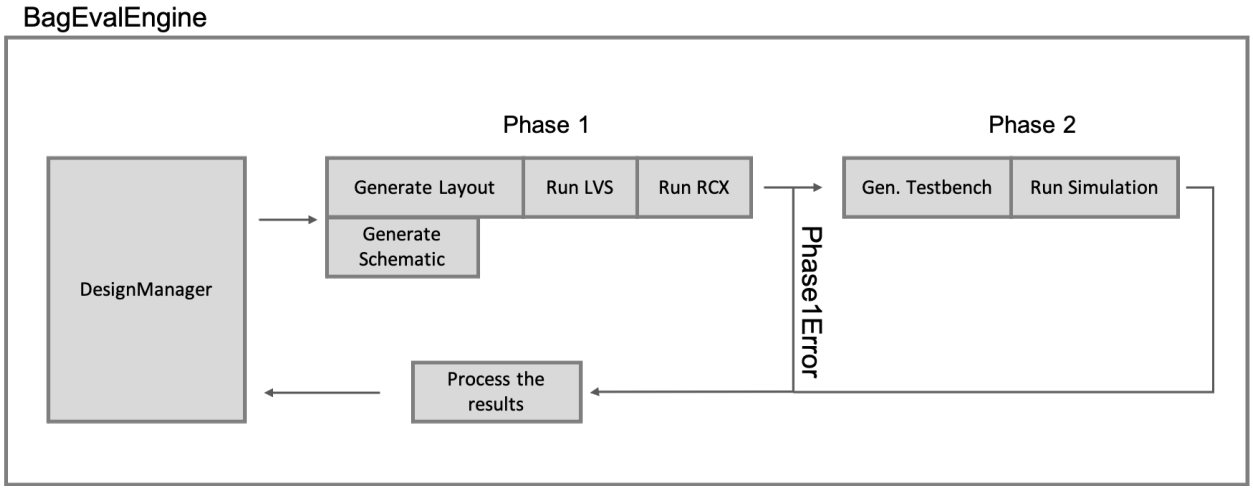


Figure 3.3: The BAG black-box engine flow.

3.6 Conclusion

In this chapter, we presented a set of simulator tools that we developed to facilitate research on analog-mixed signal circuit design automation using machine learning. We described a framework that is built on Berkeley Analog Generator [12] which allows us to formalize layout automation as a black-box optimization problem. We also presented an open-source counterpart built on `NGSPICE` which allows us to quickly iterate on algorithmic ideas and development of robust optimization algorithms without having to worry about issues with licensing or slowness of simulators.

Chapter 4

Boosting Sample Efficiency of Evolutionary Algorithms

In this chapter, we present BagNet [38] where our goal was to show that we can use neural networks to significantly reduce the number of simulations for design evaluation. If we iteratively use the generated samples during optimization to re-train the neural network we can iteratively improve the model's accuracy. A key insight is that the neural network model is not trained to the accuracy to replace the simulator, but to perform a reasonably accurate comparison between candidate design instances. We show that using this approach we can boost the sample efficiency of evolutionary algorithms to a point that we can automatically find the correct parameters of the layout for a relatively complex design of a 10 Gb/s photonic link receiver, in almost one day.

4.1 Population-Based Methods: Benefits and Drawbacks

Population-based methods have been extensively studied in the past in the application of analog circuit design automation [92, 64, 66]. These methods usually start from an initial population and iteratively derive a new population from the old one using some evolutionary operations (i.e. combination, mutation). Some selection mechanism then picks the elites of the old and new populations for the next generation, a process known as elitism. This process continues until the average cost of the current population reaches a minimum.

While this could work in principle, it is very sample inefficient, prone to instability in convergence, and requires a lot of domain-specific knowledge to hand-craft good selection mechanisms. The process must be repeated numerous times due to its stochastic nature. As a result, these methods are not suitable for layout-based optimizations where simulation takes a significant amount of time.

The sample inefficiency arises from two factors. Firstly, the majority of the new population will only slightly improve on their ancestors, and as the population improves, the

difficulty of replacing old designs increases. Therefore, it would take many iterations until the children evolve enough to surpass the average of the parents. Much of the previous work seeks to improve this by focusing on modifying the evolutionary operations such that they would increase the probability of producing better children while preserving the diversity [95]. Unfortunately, these methods have not been able to sufficiently improve the sample efficiency to accommodate the post-layout simulations. Secondly, many of these methods only look at the total cost value and do not consider the sensitivity of the cost to each design constraint, meaning that they do not account for how each specification metric is affecting the overall cost. Expert analog designers usually do this naturally by prioritizing their design objectives depending on what constraint limits their design the most. Considering only the total cost value can be misleading and may obfuscate useful information about the priority of optimizing the metrics.

To address the first issue, if we had access to an oracle that could hypothetically tell us how two designs were compared in terms of each design constraint, we could use it to direct the selection of new designs. Each time a new design is generated we can run the oracle to see how the new design compares to some average design from the previous generation. In this work, we devise a deep neural network model that can imitate the behavior of such an oracle.

To address the second problem, we can look at the current population and devise a set of critical specifications (i.e specifications that are the most limiting and should be prioritized first). In each step that we query the oracle, we only add designs that have better performance than the reference design in all metrics in the critical specification set. The important point to note is that once a metric enters the critical specification set it never becomes uncritical, as we do not want to forget which specifications derived the selection of population before the current time step. For finding the critical specification at each time step we use a heuristic which is best described by the pseudo code in algorithm 1.

Algorithm 1 Pseudo-code of the heuristic used for updating critical specification list

Input: Population buffer $\mathcal{B}$, specification list $\mathcal{S}$, critical specification list $\mathcal{CS}$ (empty at first), a reference index k (i.e 10)
if $\mathcal{CS}$.empty() **then**
    $\widetilde{\mathcal{B}} \leftarrow$ sort $\mathcal{B}$ by $cost(x) = \sum_{i \in \mathcal{S}} w_i * p_i(x)$
else
    $\widetilde{\mathcal{B}} \leftarrow$ sort $\mathcal{B}$ by $cost(x) = \sum_{i \in \mathcal{CS}} w_i * p_i(x)$
critical_spec $\leftarrow \arg\max_{i \in \mathcal{S}} \max_{x \in \widetilde{\mathcal{B}}[0:k-1]} p_i(x)$
$\mathcal{CS}.append$(critical_spec)

As an experimental exercise, we can realize the oracle with a simulator and use the aforementioned heuristics to decide whether to add a new design to the population. Figure 4.1 illustrates the expected convergence performance of this oracle compared to the same

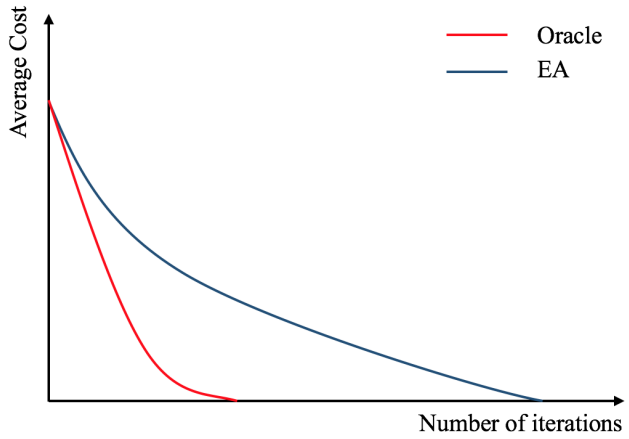


Figure 4.1: Illustration of improvement of Oracle against EA. Each iteration is equivalent to adding some number of new designs to the population. The region between the two curves represents room for improvement.

evolutionary algorithm without this discrimination. The expected behavior is that the oracle can significantly reduce the number of iterations for convergence if we knew what designs to add and what designs to reject.

However, we cannot use this oracle if we want to scale our method to do layout-level optimizations on more complex circuit topologies with larger design spaces and more expensive simulation runs. This is because the oracle has to run simulations for all generated instances to determine which designs to add or reject and therefore, there is no reduction in the number of simulations that it runs. In the next section, we propose a deep neural network model that can imitate the behavior of this oracle while significantly reducing the required number of simulation runs.

4.2 Model for Imitating the Oracle

Multiple options exist for imitating the oracle. First, we can have a regression model to predict the cost value and then use this predicted value to determine whether or not to accept a design. The cost function that the network tries to approximate can be extremely non-convex and ill-conditioned. Thus, from a limited number of samples it is very unlikely that it would generalize well to unseen data. Moreover, the cost function captures too much information from a single scalar number, so it would be hard to train.

Another option is to predict the value of each metric (i.e. gain, bandwidth, etc.). While the individual metric behavior can be smoother than the cost function, predicting the metric

value is unnecessary since we are simply attempting to predict whether a new design is superior to some other design. Therefore, instead of predicting metric values exactly, the model can take two designs and predict only which design performs better in each metric. Figure 4.2 illustrates the model architecture used for imitating the oracle.

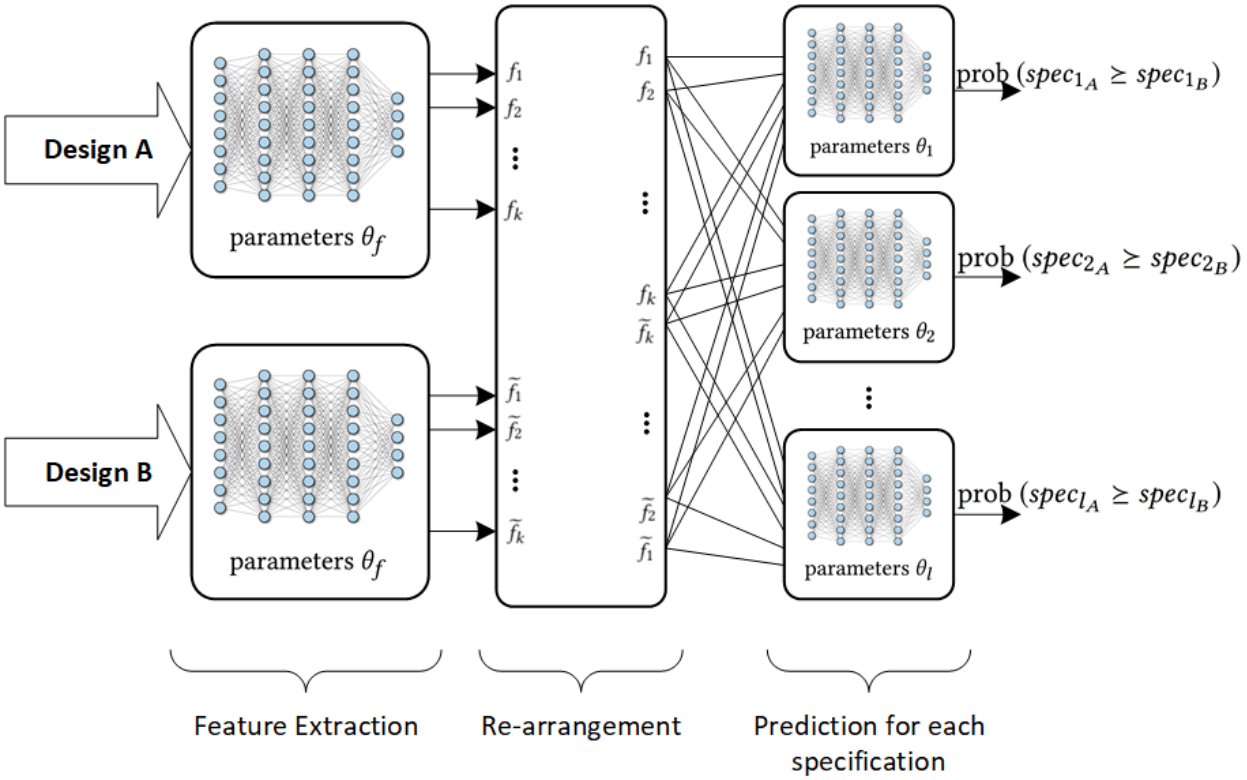


Figure 4.2: DNN's model used BagNet, $\theta = [\theta_f, \theta_1, ..., \theta_l]$ contains the parameters of the DNN. $\mathcal{M}_\theta(D_A, D_B)$ is the output probabilities of the DNN parametrized by θ for inputs D_A and D_B. $\mathcal{M}_\theta(D_A, D_B; i)$ denotes the predicted probability for the i^{th} specification. Note that $\succeq$ denotes preference, not greater than.

The model consists of a feature extraction component comprised of only fully connected layers which are the same for both Design A and Design B. For each specification, there is a sub-DNN that predicts the preference over specifications using fully connected layers. There is a constraint that the network should predict complementary probabilities for inputs $[D_A, D_B]$ vs. $[D_B, D_A]$ (i.e. $\mathcal{M}_\theta(D_A, D_B) = 1 - \mathcal{M}_\theta(D_B, D_A)$) meaning that there should be no contradiction in the predicted probabilities depending on the order by which the inputs were fed in. Therefore to ensure this property holds and to make the training easier, we can impose this inductive bias on the weight and bias matrices in the decision networks. To do so, each sub-DNN's layer should have an even number of hidden units, and the corresponding weight and bias matrices should be symmetric according to the following equations:

$$y_{m\times 1} = \mathbf{W}_{m\times 2k} x_{2k\times 1} + b_{m\times 1}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{\frac{m}{2}\times 2k} \\ \widetilde{\mathbf{W}}_{\frac{m}{2}\times 2k} \end{bmatrix} \begin{bmatrix} x_{k\times 1} \\ \widetilde{x}_{k\times 1} \end{bmatrix} + \begin{bmatrix} b_{\frac{m}{2}\times 1} \\ \widetilde{b}_{\frac{m}{2}\times 1} \end{bmatrix}$$

Where we have,

$$\widetilde{\mathbf{W}}(i,j) = \mathbf{W}(\frac{m}{2} - 1 - i, 2k - 1 - j)$$
$$\text{for } i = 0, ..., \frac{m}{2} - 1 \text{ and } j = 0, ..., 2k - 1$$
$$\widetilde{b}(i) = b(\frac{m}{2} - 1 - i) \text{ for } i = 0, ..., \frac{m}{2} - 1 \ .$$

If the weight and bias parameters are set as above, when the input order is changed from $[D_A, D_B]$ to $[D_B, D_A]$ the very first feature vector is changed from $[f_1, ..., f_k, \widetilde{f_k}, ..., \widetilde{f_1}]$ to $[\widetilde{f_1}, ..., \widetilde{f_k}, f_k, ..., f_1]$. Thus, for the last layer that has two outputs, the sigmoid function will produce $1 - \mathcal{M}_\theta(D_A, D_B)$ instead of $\mathcal{M}_\theta(D_A, D_B)$.

To train the network, we construct all permutations from the buffer of previously simulated designs and evaluate their performance in each metric. We then update network parameters with stochastic gradient descent.

4.3 Algorithm

To put everything together, Figure 4.3 illustrates the high-level architecture of the optimizer. In each iteration we use the current population and perform some specific evolutionary operations to get the next generation of the population, but we do not simply simulate them and consider them as the next generation. We use the DNN to predict if they will be better compared to some reference design already within our current population, and if the answer is positive we simulate the designs, call them the next generation, and proceed with the evolutionary algorithm.

The children may have a distribution mismatch from the data that DNN was trained on. To mitigate the distribution drift, each time we add new children we evaluate them with the actual simulator and re-train the model with correct labels so that the buffer is updated with correct labels for the next steps of training. This idea is very similar to DAgger [105] except that we do not relabel all of the children, rather we only relabel the accepted ones. Algorithm 2 shows the entire algorithm, step by step.

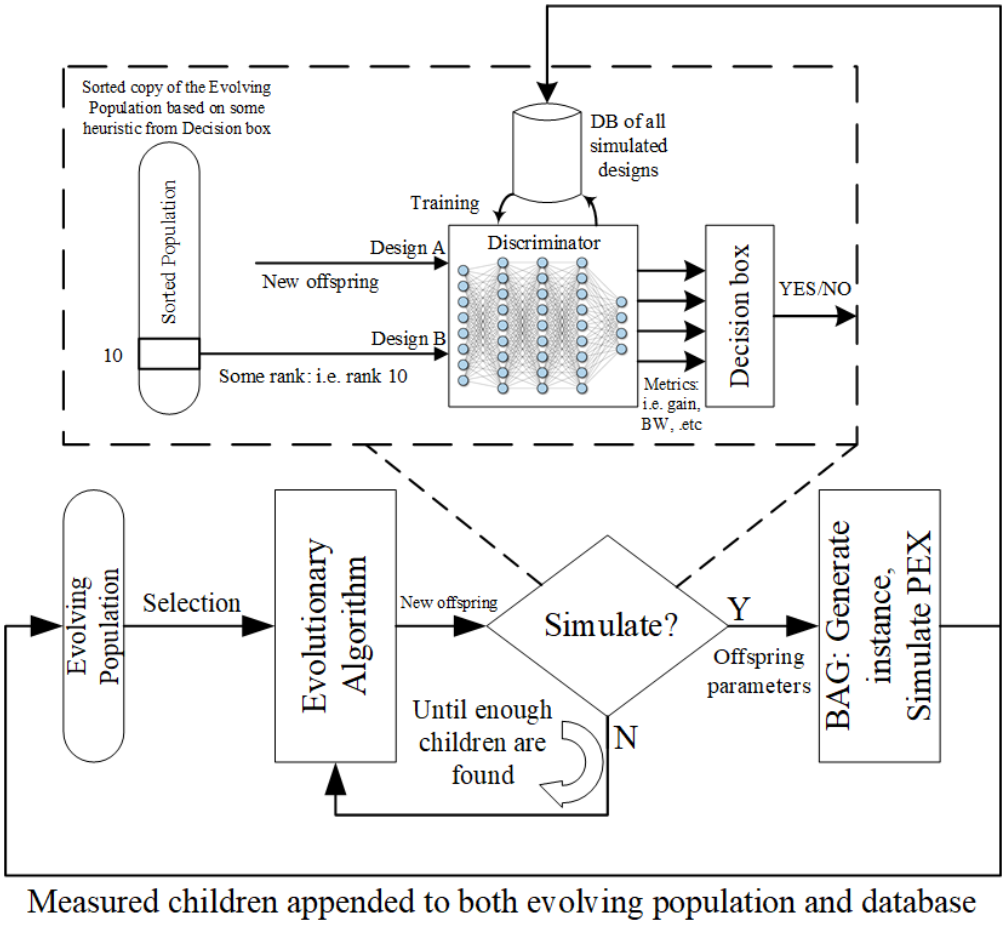


Figure 4.3: High level architecture of our optimizer

4.4 Experiments

In this section, we study a variety of experiments that clarifies some aspects of the algorithm and illustrates its capabilities on a variety of circuits.

Schematic Design of a Two Stage Amplifier

First, to clarify the convergence behavior and benefits of the algorithm, we use a simple two-stage op-amp evaluated only through schematic simulation using 45nm BSIM models on `NGSPICE`.

The circuit's schematic is shown in Figure 4.4. The objective is to find the size of transistors and the value of the compensation capacitor such that the described circuit satisfies the requirements set in table 4.1. We fixed the length and width of the unit-sized transistors to 45 nm and 0.5 μm, respectively, and for the size of each transistor, we

Algorithm 2 Pseudo-code for the entire algorithm

Given Some evolutionary operations $\mathcal{E}$ {e.g. CEM [8]}
Given Some Initial buffer of random simulated designs $\mathcal{B}$
Given reference index k {e.g. $k = 10$}
Given DNN $\mathcal{M}_\theta$ parametrized by θ
update θ {e.g. For 10 epochs}
while num_iter $<$ max_num_iter **do**
  Get critical specification list $\mathcal{CS}$ according to the heuristic {i.e. Algorithm 1}
  $\widetilde{\mathcal{B}} \leftarrow$ sort $\mathcal{B}$ by $cost(x) = \sum_{i \in \mathcal{CS}} w_i * p_i(x)$
  $\mathcal{D}_{ref} = \widetilde{B}[k]$
  list of new children $\mathcal{L} = []$
  while $\mathcal{L}$.length < 5 **do**
    $\mathcal{D}_{new} \leftarrow \mathcal{E}.generate(\mathcal{B})$ {generate a new design}
    $\mathcal{P} \leftarrow M_\theta(\mathcal{D}_{new}, \mathcal{D}_{ref})$
    if $\mathcal{P}[i] = 1,\ \forall i \in \mathcal{CS}$ **then**
      Run simulation on $\mathcal{D}_{new}$
      $\mathcal{L}$.append($\mathcal{D}_{new}$)
    else
      Continue
  $\mathcal{B} \leftarrow \mathcal{E}$.select($\mathcal{B} + \mathcal{L}$) {Elitism}
  update θ {e.g. For 10 epochs}

limit the number of fingers to any integer number between 1 to 100. For compensation we also let the algorithm choose C_c from any number between $0.1p$F to $10p$F with steps of $0.1f$F. The grid size of the search space is 10^{14}. A given instance is evaluated through DC, AC, CMRR, PSRR, and transient simulations, which in total takes approximately one second for each design. Therefore, brute-force sweeping is not practical even in this simple example. However, this short simulation time allows us to do comparisons against the vanilla evolutionary algorithm and the oracle. Note that these methods are not feasible for layout-based simulations, however, since each would take several minutes per design to evaluate instead of a second.

Table 4.1 shows the performance of the minimum cost solution found by different approaches. In this example, all approaches found a solution satisfying all specifications, but this is not guaranteed as it depends on the feasibility of the specifications and also the stochastic behavior of the evolutionary algorithms. We can always adjust exploration vs. exploitation of the evolutionary algorithms by the mutation rate, but this will increase the convergence time of all of the approaches. However, our approach will not need as many simulations as the oracle or the vanilla evolutionary algorithms.

To avoid over-fitting and being certain about false positives and negatives, we can leverage Bayesian DNNs within our model [79], which can estimate the uncertainty regarding the

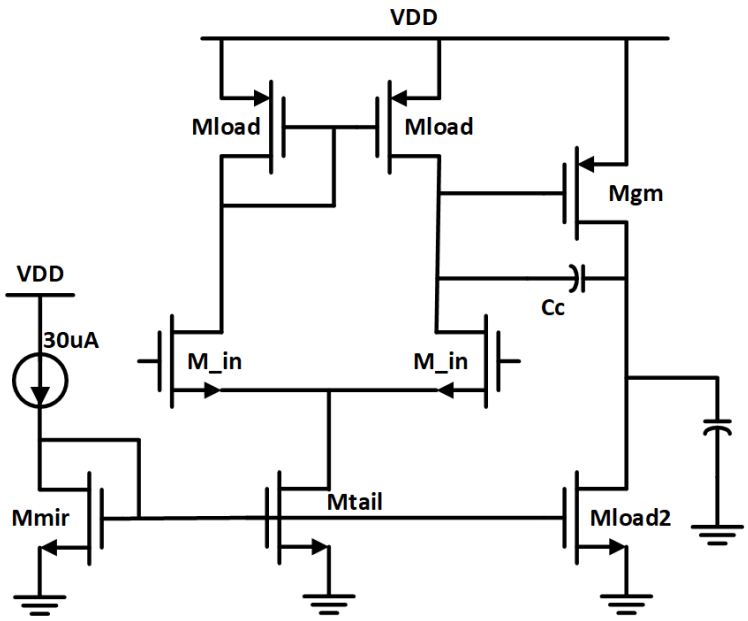


Figure 4.4: Schematic of a vanilla two stage op-amp

Table 4.1: Objective of design and performance of solutions found using different approaches for the two stage op-amp example

	Requirement	Evolutionary	Oracle	**Ours**
Gain	>300	323	314	**335**
f_{unity} [MHz]	>10	10.83	10.66	**10.2**
Phase Margin [°]	>60	60.7	60.83	**62**
$t_{settling}$ [ns]	<90	59.9	83.5	**62**
CMRR [dB]	>50	53	54	**54**
PSRR [dB]	>50	57	56	**57**
Systematic Offset [mV]	<1	0.823	0.94	**0.32**
Ibias [μA]	<200	188	158	**148**

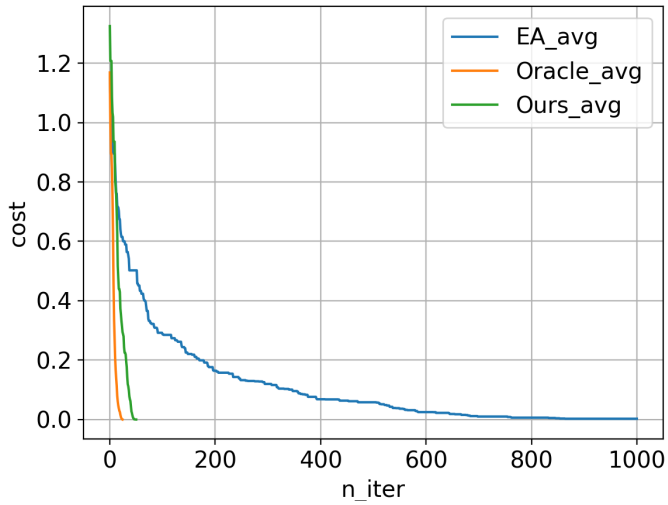
Table 4.2: Summary of number of operations involved in the process of each approach

	# of NN Queries	# of Re-training	# of Simulations
Simple Evolution	-	-	5424
Oracle	-	-	3474
Ours	55102	50	241

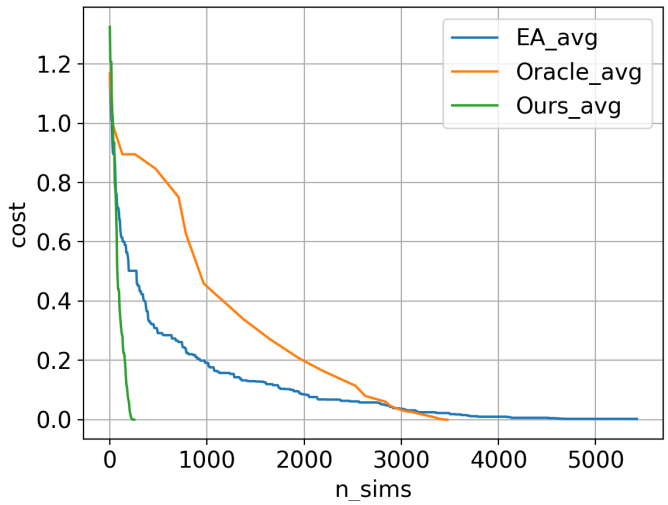
decisions. In this specific example we use drop-out layers which can be considered as Bayesian DNNs with Bernoulli distributions [30]. During inference, we sample the model 5 times and average the probabilities to reduce the uncertainty about the decisions.

Figure 4.5a shows the average cost of the top 20 designs in the population for the oracle, vanilla evolutionary, and our algorithm. Each time only 5 designs are added and the evolutionary operations are the same for all experiments. We ran our approach on multiple random initial seeds to ensure robustness in training and performance. We note that in terms of the number of iterations, the oracle requires the least iterations, but it runs a simulation on all generated designs to make selection decisions so this does not translate to the overall fastest time. Therefore, it is impractical to run it on post-layout simulation on more complex circuits. Figure 4.5b shows performance in terms of the number of simulations. We see that our approach is more efficient by at least a factor of 10 in this simple example.

Table 4.2 shows a summary of number of operations in our simple example. We note that our approach can cut down a lot of impractical simulations at the cost of more time spent on training and inference of a DNN. With recent advancements in hardware for machine learning and the use of GPUs, the time spent on training and inference can be significantly reduced. When we want to scale up to more intricate circuits two factors make our approach advantageous. Firstly, when we do layout optimization, simulation drastically increases proportionally to the circuit size. Also, more complicated circuits have larger design space and it will become even more critical to prune out impractical regions of design space as we scale up. Therefore, in terms of scaling to layout optimization, our approach seems to be promising.



(a)



(b)

Figure 4.5: a) Average cost of top 20 individuals across number of iterations. Each iteration corresponds to adding 5 designs to the population. b) Average cost of top 20 individuals across number of simulations.

Layout Design of a Two-Stage Amplifier and Comparison Against an Expert Design

This example is presented to compare an expert-designed circuit with our algorithm's design. The op-amp's topology and the cardinality of search space are shown in Figure 4.6, with each array denoting how many parameters were considered for design. For example for Mref, 20 values of n_fingers were considered. In total, this design example has an 11-dimensional exploration space with a size of 3×10^{13}.

The topology is a standard Miller compensated op-amp, in which the first stage contains diode-connected and negative-gm loads. The design procedure is more cumbersome than the previous two-stage example, mainly because of the positive feedback. A scripted design procedure for this topology is included as part of BAG to exemplify codifying expert-driven design methodologies. The design script can find the proper transistor sizing while considering layout parasitic effects using a closed-loop design methodology. The inputs to the design script are specifications of phase margin and bandwidth, and the objective is to maximize gain. In this circuit, the resistor and capacitor are schematic parameters, while all transistors and all connecting wires use the GF14 nm PDK extraction model.

Table 4.3 shows a performance summary of our approach compared to that of the design generated by the design script. The script is unable to meet the gain requirement due to a designer-imposed constraint that the negative g_m should not cancel more than 70% of the total positive resistance at the first stage's output. This constraint arises from a practical assumption that there will be a mismatch between the negative g_m's resistance and the overall positive resistance, due to process, voltage, and temperature (PVT) variations. Thus, the circuit can become unintentionally unstable, and therefore during design, we leave some margin to accommodate these prospective random variations. We have the option of imposing a similar constraint to equate the design spaces, or we can run simulations over process and temperature variations to ensure that our practical constraints are not too pessimistic.

For our approach, the initial random population size is 100 with the best cost of 0.3. We ran every simulation on different PVT variations and recorded the worst metric as the overall performance value. Figure 4.7 illustrates how the average cost in top 20 designs changes over time. Reaching a solution with our approach took 3 hours including initial population characterization, whereas developing the design script takes 4-7 days according to the expert. The DNN was queried 3117 times in total (equivalent to 6 minutes of run time on our compute servers) and we only ran 120 new simulations in addition to the initial population of size 100 (each of which takes on average 48 seconds to run). Moreover, the complexity of developing a design script forces the designer to limit the search space to make the process feasible and a generic design algorithm that properly imposes high-level specifications onto a large system is immensely difficult to generate. We will see an example of such systems and our approach's solution in the next section.

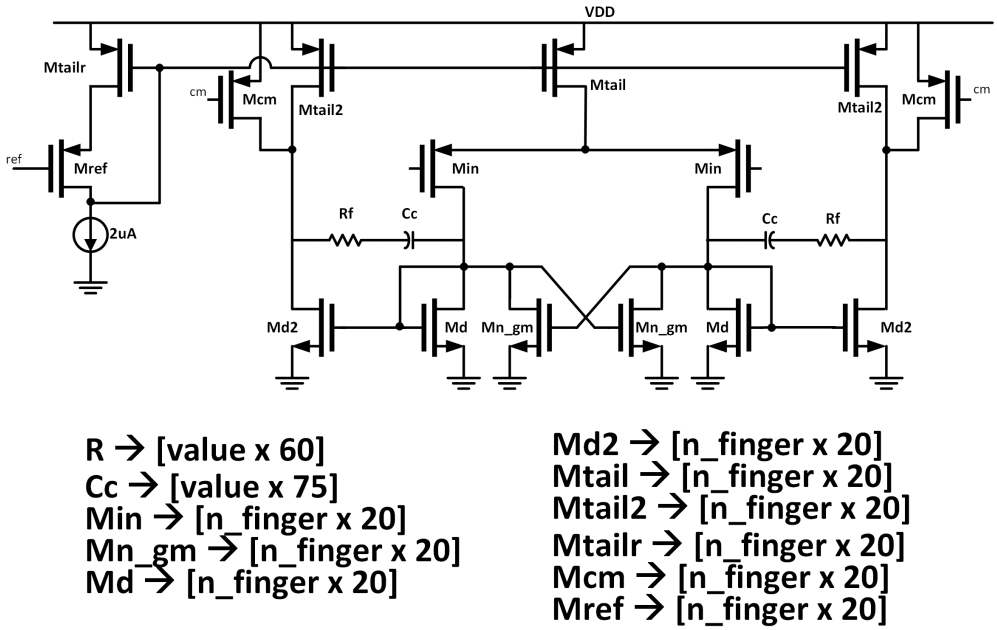


Figure 4.6: Two stage op-amp with negative g_m load

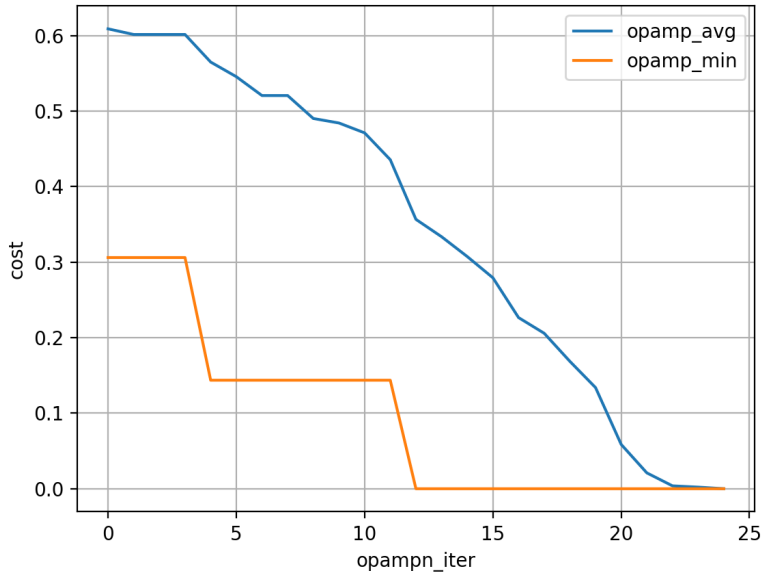


Figure 4.7: Convergence curve of the two stage op-amp done in BAG. Orange shows the minimum cost across iteration step and blue shows the average of cost in top 20 instances

Table 4.3: Performance of expert design methodology and our approach

	Requirement	Expert	Ours
f_{unity}	>100 MHz	382 MHz	159 MHz
pm	$> 60°$	64°	75°
gain	>100 (for ours)	42	105

End-to-End Layout Optimization of an Optical Receiver Link

The following experiment highlights the capabilities of our approach in handling complex analog-mixed signal design problems using post-layout simulations. This problem has 26 different parameters to set, which makes the cardinality of the design space 2.8×10^30 (compared to 10^16 for the two-stage opamp). We demonstrate a differential optical link receiver front-end with a single tap double tail sense amplifier (DTSA) in the end. The circuit is shown in Figure 4.8 with design space parameters at the bottom. The goal is to design this circuit from very high-level specifications, namely, data rate, power consumption, and minimum sensitivity for a given bit error rate (BER).

Automating the characterization of instances of this circuit is the key to setting up the environment before running the algorithm. The following steps are crucial to get performance metrics on each design. First, we instantiate the DTSA's layout, schematic, and extracted netlist. We then run overdrive test recovery to characterize the transient behavior of DTSA. Figure 4.9a shows a typical overdrive test recovery curve for a given comparator. We specifically measure *vcharge*, *vreset*, and *vout* in the time instances relative to the edge of the clock as shown in the figure. By specifying these three numbers as well as a minimum *vin* (i.e 1 mV), we can describe the performance of the comparator at a given data rate. To get the noise behavior of the comparator, we run numerous transient noise simulations for several cycles while sweeping input voltage from a small negative voltage to a small positive voltage. We can then fit a normal Gaussian distribution to the estimated probability of ones in each transient run and get an estimation of the input-referred voltage noise of the DTSA. Figure 4.9b illustrates this simulation procedure. We then take the entire system's extracted netlist and characterize the behavior of the analog front end (AFE) while the DTSA is acting as a load for the continuous-time linear equalizer (CTLE).

Once we used noise simulations to get the input-referred voltage noise of the comparator, we can then aggregate the comparator's noise from previous simulations with the AFE's noise to compute the total root mean square noise at the comparator's input. We also use the transient response of the circuit to ensure that the eye diagram has high fidelity at the input of the comparator. The input-output curves, and formulas used to measure eye's fidelity are shown in figure 4.10a and 4.10b.

We estimate eye height and eye thickness ratio and specify a constraint on them to describe the quality of the eye diagram for a given input sensitivity. Using BER of 10^{-12}

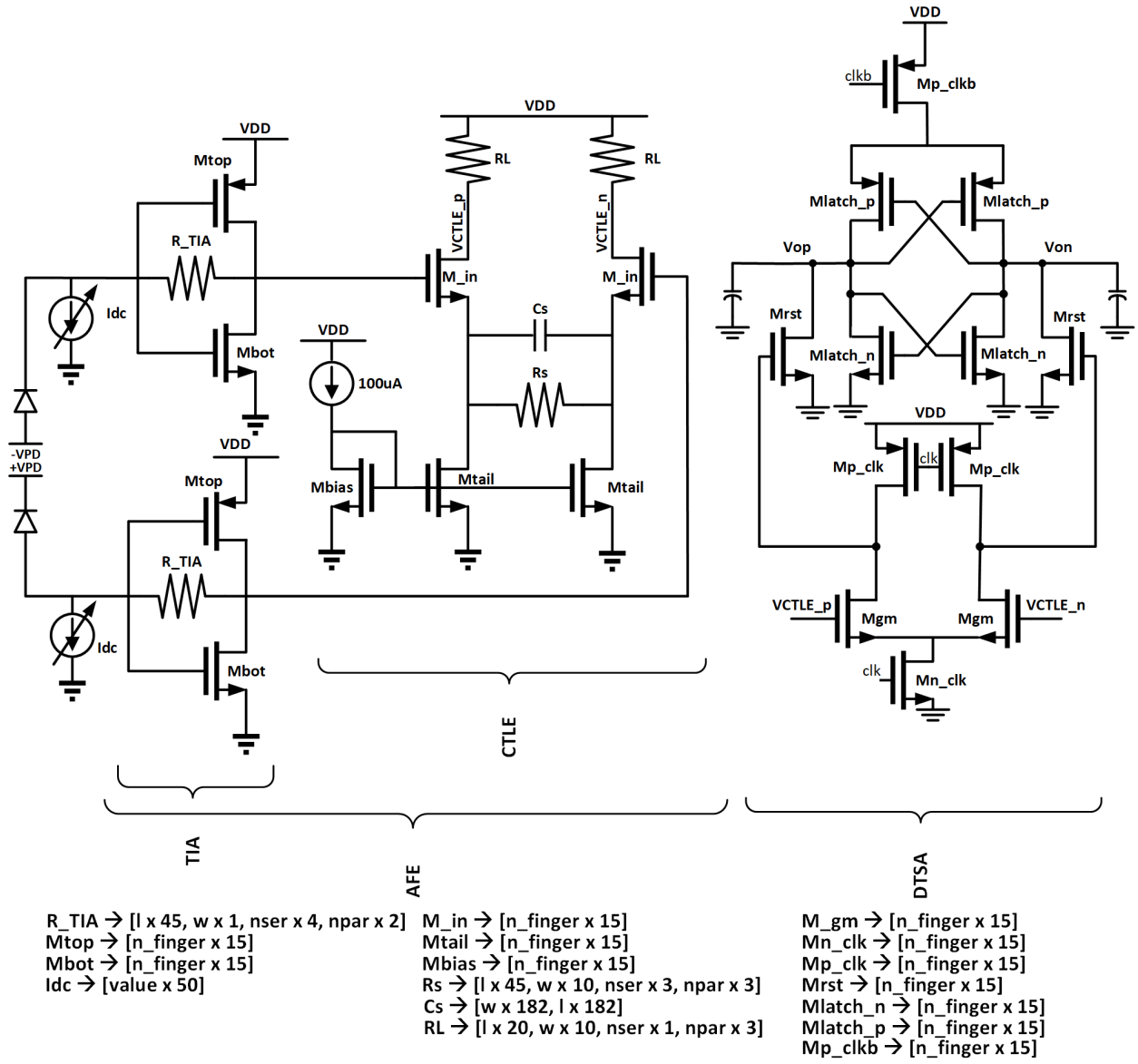
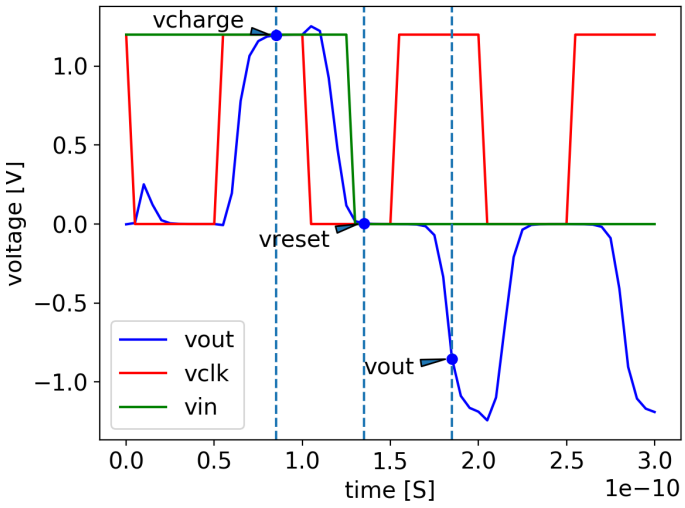
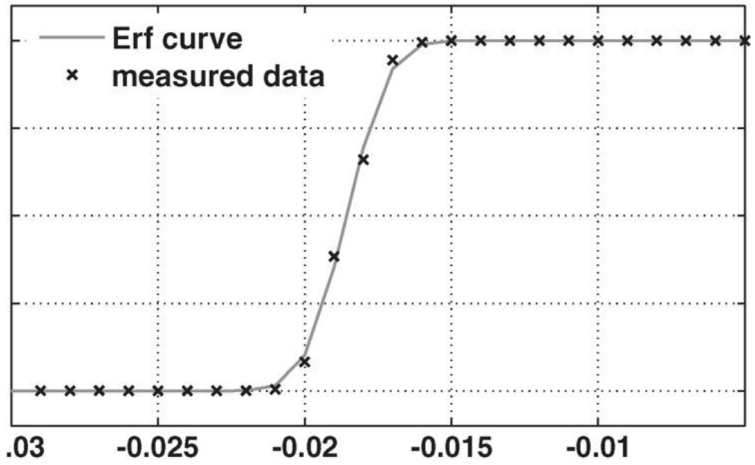


Figure 4.8: Optical receiver schematic

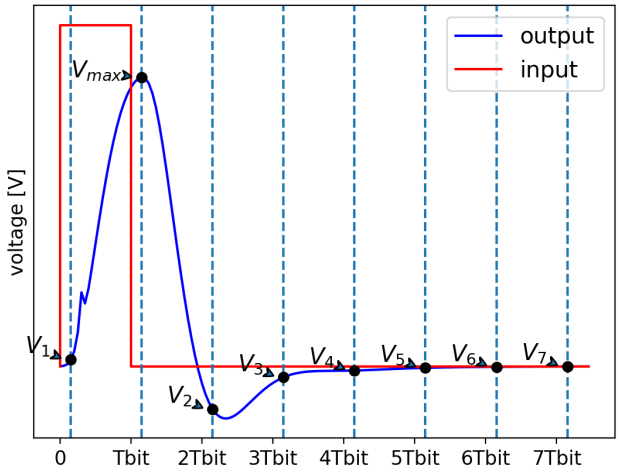


(a)

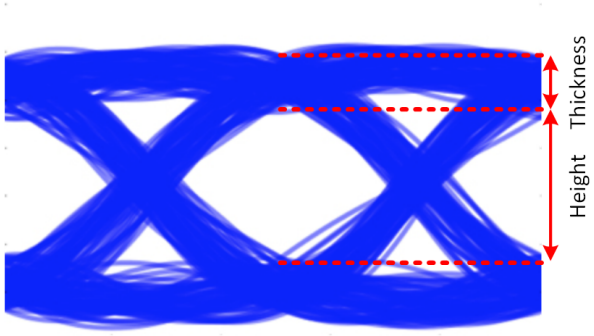


(b)

Figure 4.9: a) Overdrive test recovery simulation curves for DTSA b) Probability of outputting a one vs. V_{in}. We can use the cumulative density function of a Gaussian to estimate the standard deviation of the noise



(a)



$$eye\_height = V_{max} - \sum_{i=1}^{\infty} |V_i|$$

$$thickness = \sum_{i=1}^{\infty} |V_i|$$

(b)

Figure 4.10: a) Input and output signals for measuring the eye height and thickness, the input is a small signal pulse with an amplitude of target input sensitivity, and with a width of T_{bit} for the target data rate. The output curve is sampled at time instances shown relative to the main cursor (maximum of output) b) Equations used for estimating eye height and thickness to express the fidelity of eye diagram

we can compute the required eye height at the input of the comparator using equation 4.1 and compare it against the actual eye height. For the optimization objective, we can put a constraint on the relative difference between the actual eye height and the required eye height (i.e. actual eye height should be 10% larger than the minimum required eye height). We call this percentage eye margin.

$$eye\_h_{min} = 9\sigma_{noise} + \text{Residual Offset} + \text{DTSA sensitivity} \tag{4.1}$$

For sensitivity, we use the V_{in} from the overdrive test recovery in previous tests. There will also be a component mismatch offset which can be significantly reduced with a systematic offset cancellation scheme. The offset cannot, however, be fully eliminated, so the residual offset will also be considered (e.g. 1mV). We also run a common-mode AC simulation to ensure that the tail transistors providing bias currents are operating in saturation by specifying a reasonable minimum common-mode rejection ratio requirement. For one instance, this whole process takes about 200 seconds on our compute servers. We can then compute the cost of each design as specified by equation 3.1. Therefore, being sample efficient in terms of learning and optimizing is vital, as discussed in section 4.4.

In terms of layout generator search space, each resistance drawn in Figure 4.8 has unit length, unit width, number of series units, and number of parallel units. The CTLE's capacitor has width and length, and each transistor has several fins and fingers that need to be determined. We fixed the number of fins to simplify the search space. The cardinality of each design parameter is written in Figure 4.8. In total, the design example has a 26-dimensional exploration space with a size of 2.8×10^{30}.

Figure 4.11 and Table 4.4 show the layout and performance of the solution for designs found with cost of 0 (satisfying all specs), 0.1, and 0.2, respectively. The first design solution was found using 435 simulations equivalent to 27 hours of run time. This number includes generating the initial population which consisted of 150 designs with the best cost function of 2.5. During the process, the DNN was queried 77487 times in total and only 285 of those were simulated, representing around 300x sample compression efficiency. From the total runtime, 1.6 hours were spent on training, and almost 2.1 hours were spent on querying the DNN.

4.5 Conclusion

In this chapter, we introduced BagNet, a sample-efficient evolutionary-based optimization algorithm for designing analog circuits using analog layout generators. In this approach, we used DNNs to prune out the useless part of the design space, so that we can save time on long simulations. We showed that the algorithm can be used in designing a variety of real, practical AMS circuits with different applications regardless of size and complexity, as long as the verification procedure is properly defined.

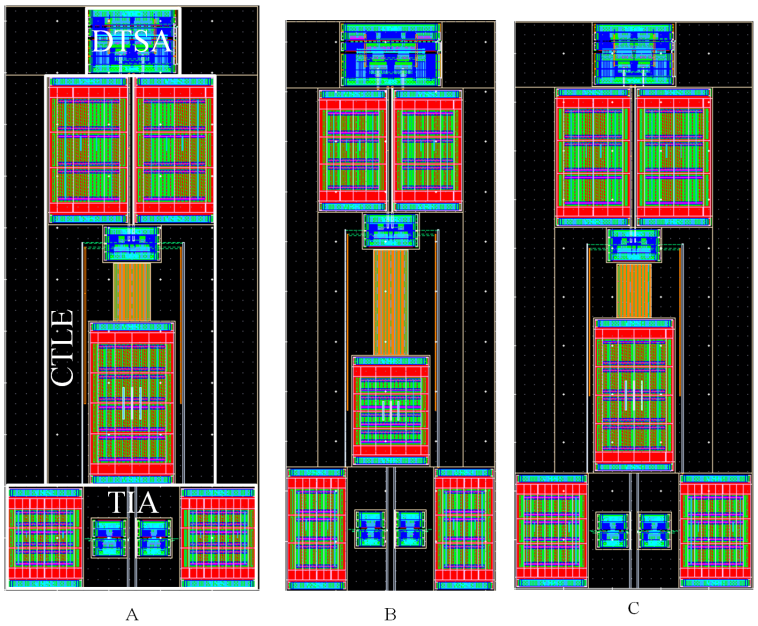


Figure 4.11: Sample layouts of the optical link receiver circuit with different cost values of 0, 0.1, and 0.2 for A, B, and C, respectively.

Table 4.4: Design Performance for optical receiver design for $C_{PD} = 20fF$, $I_{min} = 3\mu A$, Data Rate $= 10Gbit/s$

	Requirement	A (cost = 0)	B (cost = 0.1)	C (cost = 0.2)
Thickness ratio	<10%	6%	3.3%	8.5%
Eye margin @ I_{min}	>10%	10.2%	15.1%	10.4%
CMRR	>3	4.77	5.51	4.5
vcharge	>0.95VDD	VDD	VDD	VDD
vreset	<1 mV	20 μV	52 μV	21 μV
vout	<-0.9VDD	-0.91VDD	-0.88VDD	-0.87VDD
Total Noise	<5 mV	2.9 mV	3.2 mV	3.05 mV
Total Ibias	<10 mA	6.2 mA	4.04 mA	6.3 mA

Chapter 5

Improving Circuit Models via Pre-training and Fine-tuning

Being able to predict the performance of circuits without running expensive simulations is a desired capability that can catalyze automated design. In what we have covered so far, the neural network model that we learn during optimization is tailored particularly to a given topology for modeling specific design metrics. If during the design process, the user suddenly decides to change the topology or change the optimization criteria the modeling has to be done from scratch and prior knowledge cannot be transferred to the new problem.

In this chapter, we present a supervised pretraining approach to learning circuit representations that can be adapted to new circuit topologies or unseen prediction tasks. We hypothesize that if we train a neural network (NN) that can predict the output DC voltages of a wide range of circuit instances it will be forced to learn generalizable knowledge about the role of each circuit element and how they interact with each other. The dataset for this supervised learning objective can be easily collected at scale since the required DC simulation to get ground truth labels is relatively cheap. This representation would then be helpful for few-shot generalization to unseen circuit metrics that require more time-consuming simulations for obtaining the ground-truth labels. To cope with the variable topological structure of different circuits we describe each circuit as a graph and use graph neural networks (GNNs) to learn node embeddings. We show that pretraining GNNs on the prediction of output node voltages can encourage learning representations that can be adapted to new unseen topologies or prediction of new circuit-level properties with up to 10x more sample efficiency compared to a randomly initialized model. We further show that we can improve the sample efficiency of the presented BagNet from chapter 4 by 2x (almost as good as using an oracle model) via fine-tuning pre-trained GNNs as the feature extractor of the learned models.

5.1 Related Work

Modeling circuits using deep neural networks. Functional modeling of analog circuits using deep neural networks has been revisited in recent years due to the success of deep learning in modeling black-box functions. To this end, a conventional modeling approach is to model the entire circuit end-to-end with multi-layer perceptrons [134, 70]. However, this choice of architecture requires a large training dataset for building accurate models which can be problematic when the simulation is time-consuming. One way to address this issue has been to impose domain-specific inductive biases on model architectures. For example, [45] proposes breaking the circuit into sub-circuits and modeling each with a separate NN in an auto-regressive fashion. However, it requires ad-hoc human input on the definition of sub-circuits. Also, the sub-circuits themselves are modeled with MLPs and do not leverage structural information. In this work, we utilize GNNs, which generalize the idea of imposing structural inductive bias on the architecture.

GNNs in chip design. In recent years, the idea of using GNNs to automate the design process of systems-on-chip has been visited in several prior works from both an optimization and modeling perspective. Most notably [76] studies the problem of chip placement and utilizes GNNs as the backbone of the feature embeddings of an RL policy and value function that is learned in an end-to-end fashion using custom reward functions. The reason behind using GNNs as feature extractors is that many chip placement strategies are motivated by the *global* structure of the system and GNNs can capture the structural dependencies effectively. [104], on the other hand, uses GNNs for circuit parasitic prediction based on the circuit's graph which only needs *local* information.

[73, 136] employ GNNs as the backbone of the predictive models for optimizing analog circuit metrics. However, the benefits of using GNNs in these works are limited to a single topology or a single optimization objective. In our work, we study pretraining GNNs to see whether we can reliably transfer domain knowledge to new topologies or prediction tasks instead of learning from scratch.

5.2 Learning representations for analog circuits

We will first present the language for describing analog circuits as graphs in section 5.2. Then we describe how we can use a supervised node property prediction task (i.e. predicting voltages of each node) as a pretraining objective to learn representations that can be fine-tuned for predicting voltages on unseen topologies or for new graph property prediction tasks (e.g. predicting the output resistance of the circuit).

New graph representation

Despite the recent surge of interest in utilizing GNNs for predictive models on circuits [104, 74], there is still no unified descriptive language for representing transistor-based circuits as

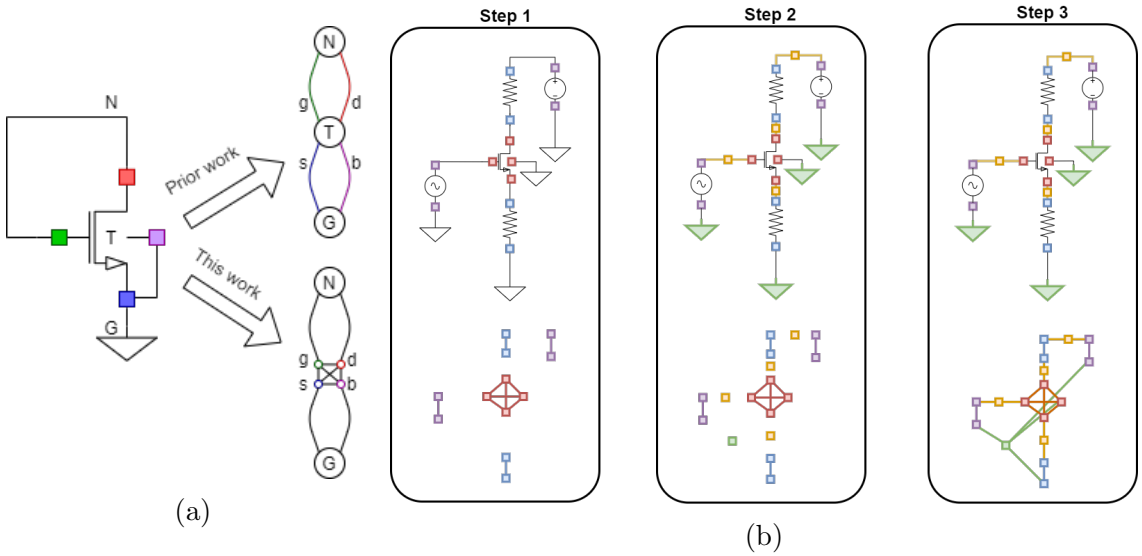


Figure 5.1: (a) Example of why it is important to explicitly model device terminals as nodes to disambiguate certain topological configurations. (b) Steps of converting a circuit schematic to a graph representation. (1) Convert all devices to their corresponding complete sub-graphs (2) Convert all circuit nets to (yellow) nodes in the graph (3) Make edge connections according to the topology of the circuit.

graphs that can uniquely represent all possible topological configurations. For example, [104] proposes to map each device and each net (i.e. wire connection) in the circuit to a graph node and uses edge types to distinguish the device terminal types (i.e. source, drain, gate, body). This representation becomes ambiguous in cases where a device (e.g. transistor) has the same net connection between two or more terminals. Figure 5.1a illustrates such an example. Since the drain (red) and gate (green) terminals of the transistor are both connected to net N, two edges with different types should connect the device T to its corresponding net N. The source (blue) and body (purple) terminals also have the same ambiguity.

To disambiguate this representation, we propose a set of steps that explicitly represent the device terminals as individual nodes. This representation is closely linked to the widely used textual description (i.e. netlist) of circuits used for simulator tools such as Spice [96]. In this representation, each device is a *complete* sub-graph with its terminals as individual nodes. The features of each node indicate the terminal type and device parameters and therefore, all edges become feature-less. Figure 5.1b illustrates an example of how a simple amplifier circuit gets mapped to its graph representation, step by step following our approach:

1. Convert the terminals of each device to a complete sub-graph with the same number of nodes as the number of terminals.
2. Convert each circuit net to a node in the graph.
3. Connect all the nets to their corresponding device terminals.

The resulting representation is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprised of a set of nodes $\mathcal{V}$ and a set of undirected edges $\mathcal{E}$. Each node is associated with a feature vector which describes the type of the node as well as the device parameters that the node belongs to (e.g. the value of a resistor). There are no features associated with edges, and they only preserve the connectivity information of the elements of the circuit.

Voltage prediction as the pretraining task

One of the major steps for analyzing and synthesizing circuits for humans is to compute the bias first. This entails computing the DC voltage of various important nodes in the circuit that sets the working condition of all the devices. One can think of computing voltages as deriving the operating point and linearizing a nonlinear system which is a pre-requisite step for computing other system metrics.

Inspired by this causal abstraction, we hypothesize that if a neural network is trained to predict the DC voltages of circuit nodes in various contexts it should be able to learn critical system-specific knowledge that can be re-used for other predictions, either in zero-shot or few-shot manner via fine-tuning. In other words, the pre-trained backbone on the voltage prediction task should provide a better initialization for unseen tasks and topologies than starting from random initialization. Moreover, getting DC operating points of circuits through simulation is relatively inexpensive compared to simulating other circuit properties. Therefore, in principle, the pre-training dataset can be collected at scale through simulation in a reasonable time frame.

Figure 5.2 illustrates the pretraining architecture. We cascade a GNN backbone with an MLP with shared parameters across nodes which predicts the output voltage based on the node contextualized representations. The pretraining objective is minimizing the mean square error (MSE) loss between the predicted output and the ground truth for the output nodes. There is no supervision signal on the representation of other non-output nodes. After pretraining, we can use the output node features from the GNN backbone for the prediction of new tasks or new topologies.

From node embedding to graph property prediction

To perform a graph property prediction task, we need to combine the node embeddings into a single graph embedding. A naive way is to aggregate all the node embeddings via pooling operations (e.g. mean pooling) and then feed it to an MLP that performs the graph level prediction. However, this approach can result in loss of information during aggregation and decreased capacity of the neural network.

To mitigate this problem, we propose a learnable attention-based aggregation layer that scales linearly with the number of output nodes. By learning the aggregation module, the model can choose the optimal way of aggregating the embeddings into a graph representation based on the dataset.

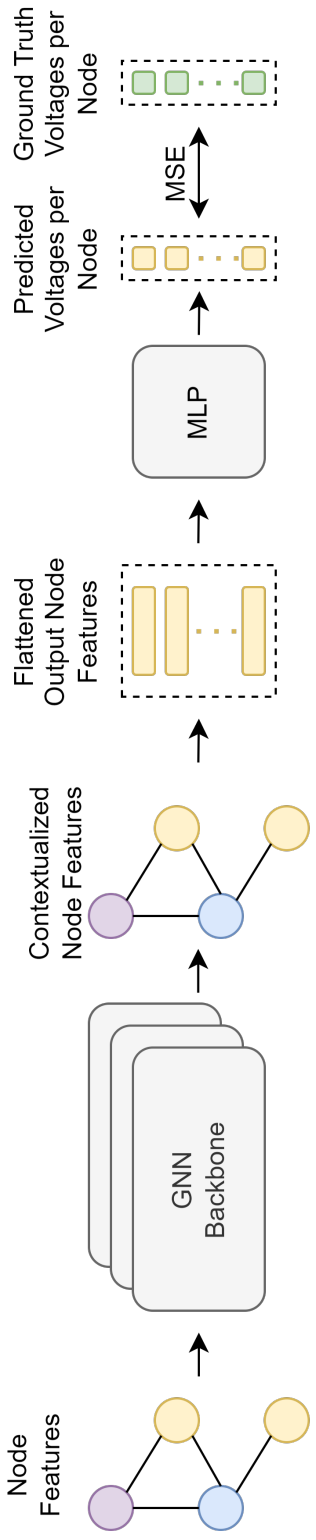


Figure 5.2: Pretraining architecture: After the GNN backbone, the batch of output node features (yellow nodes) are fed into an MLP to predict the output voltages of each node. The same MLP is applied to all the node features. The gradient of the mean square error between ground truth and prediction is then backpropagated to update the parameters.

Figure 5.3 demonstrates this architecture for fine-tuning on a graph property prediction task. Once we have the contextualized node embeddings at the output of the GNN, we flatten them as a *set* of feature vectors and pass them through a cross attention network. In this attention module, we compute the cross attention between a set of learned latent vectors and the output node features to embed them into a single graph representation. The latent vectors are represented as a 16x16 array that is learned during training to minimize the output MSE loss. The contextualized 16x16 array of latent vectors is then flattened to a single vector of size 256 at the output to represent the graph embedding. This architecture is inspired by the transformer read-out layer proposed in [55] but modified to be similar to perceiver IO's cross attention architecture [54] to gracefully scale computation and memory with the number of nodes. [1]

Evaluation Metric

The prediction tasks presented in this paper are all regression tasks that require minimizing a mean square error loss of the predicted values w.r.t. the ground truth. MSE however, is notoriously sensitive to outliers and sensitive to the range of the predicted output. Therefore, to get a better quantitative measure of performance, we propose measuring the accuracy of correct predictions within a certain resolution.

For a given dataset $\{x_i, y_i\}_{i=1}^{N}$ let $\hat{y}_i$ be the predictions of the model. We define Acc@K as

$$\text{Acc@K} = \frac{1}{N}\sum_{j=1}^{N} \mathbb{1}(|y_j - \hat{y}_j| \leq \frac{\max_i y_i - \min_i y_i}{K}). \tag{5.1}$$

This metric measures the ratio of predictions that are within a certain precision w.r.t to their ground truth value. For instance in the voltage prediction task (with a range of 1V), $K = 100$ sets the precision to $10mV$, i.e. 1% of the range.

5.3 Experiments

We perform an extensive empirical study to validate our hypothesis by focusing on the following questions: (1) Can we successfully pre-train GNN architectures to regress the DC voltage of output nodes on a variety of circuits? (2) How does a pre-trained network perform on predicting the output voltage nodes of an unseen circuit topology that is slightly different than what it has been pre-trained on? (3) Can we fine-tune the pre-trained backbone to improve voltage prediction generalization on new unseen circuits? (4) Can we re-use the

[1] The regular transformer read-out layer proposed in [55] has a memory and computation footprint of $O(N^2)$ (where N is the number of nodes), but the cross attention layer proposed in [54] (and ours) scales linearly with N.

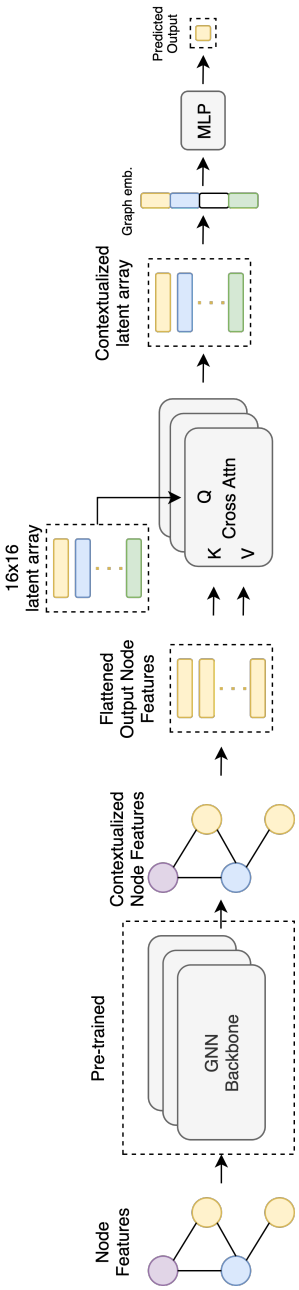


Figure 5.3: Node to graph prediction pipeline: After the GNN backbone, the node features are translated to a single vector representing the graph embedding. The graph embedding is then fed to an MLP to predict the output. The node to graph embedding is 3 cross attention layers between a learned embedding and the node features (similar to [54])

learned representations after pre-training, to efficiently learn to predict new unseen circuit-level tasks (e.g. a design metric of a circuit) via fine-tuning? (5) Can we improve optimization sample efficiency of model-based circuit optimization tasks like BagNet (Chapter 4)?

In these experiments, all the models are pre-trained by minimizing the mean square error loss on the predicted outputs using the Adam optimizer and learning rate with linear decay from $1e-3$ to $5e-5$. The other common hyper-parameters are batch size of 256, activation of Relu, and hidden channel size of 128 for GNNs and 512 for MLPs. The longest training run-time belongs to running DeepGEN-15 which approximately takes $\sim$ 5 hrs on a single NVIDIA GeForce GTX TITAN X.

The code for datasets, data loaders, and evaluators are released under BSD 3-Clause license at `https://github.com/kouroshHakha/circuit-fewshot-code`. The data loaders are compatible with Pytorch Geometric [26] interface. We provide automatic dataset downloading, processing, and evaluation, as well as the implementation of example baselines discussed in the paper. We have a separate repository for data generation through circuit simulators that can be found at `https://github.com/kouroshHakha/circuit-fewshot-data`. The raw datasets generated from this code-base are automatically downloaded when the previous code-base is utilized. The code for optimization experiments is found under `bagnet_gnn` branch of the existing repository of BagNet at `https://github.com/kouroshHakha/bagnet_ngspice/tree/bagnet_gnn`.

Dataset and circuits under study

We perform our experiments on two sets of circuits. The first one is a family of resistor-based circuits that allow us more control over different generalization aspects of the problem. The second one is different variations of a real-world transistor-based two-stage operational amplifier (OpAmp) circuit that illustrates the generalization of our method to more real-world examples where modeling design constraints is challenging.

Resistor-based circuits. For pre-training we chose a particular topological structure made only of resistors and voltage sources known as resistor ladders (Figure 5.4). For data for pre-training, we can easily vary the number of ladder branches, values of resistors, and voltage sources to cover enough support for resistors and voltage sources in various contextualized neighborhoods in the graphs. During test time, we can see if the model has transferable domain knowledge relevant to resistor-based circuits by evaluating resistor ladders with more branches or new unseen resistor-voltage source topological structures. To this end, we generate 20k training instances of resistor ladders with 2 to 10 branches with equal distribution weight for each. For each node in each circuit, we solve circuit equations to get the ground truth for voltage values.

Operational Amplifier (OpAmp). This family of circuits has been studied in many prior works in circuit design optimization and modeling including the experiments we discussed in chapter 4. For pre-training, we consider another variation of this circuit, shown in Figure 5.5. We generate 126 thousand design instances by varying device parameters, voltage and current source values, and the place that circuit input is applied (i.e. differential input,

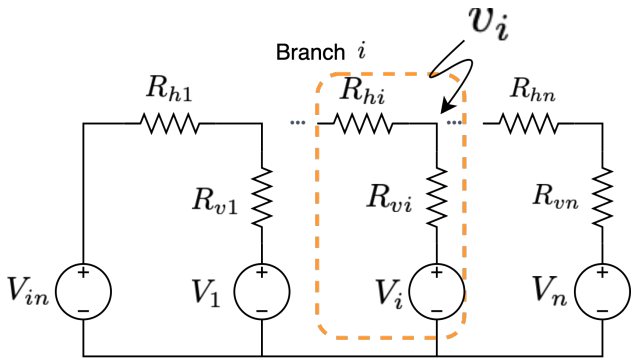


Figure 5.4: The circuit schematic of the resistor ladder. This circuit is comprised of n branches. The pretraining task is to predict the output voltage of each branch v_i. A resistor ladder with n branches has $2n$ resistors, $n+1$ voltage sources, and n output nodes.

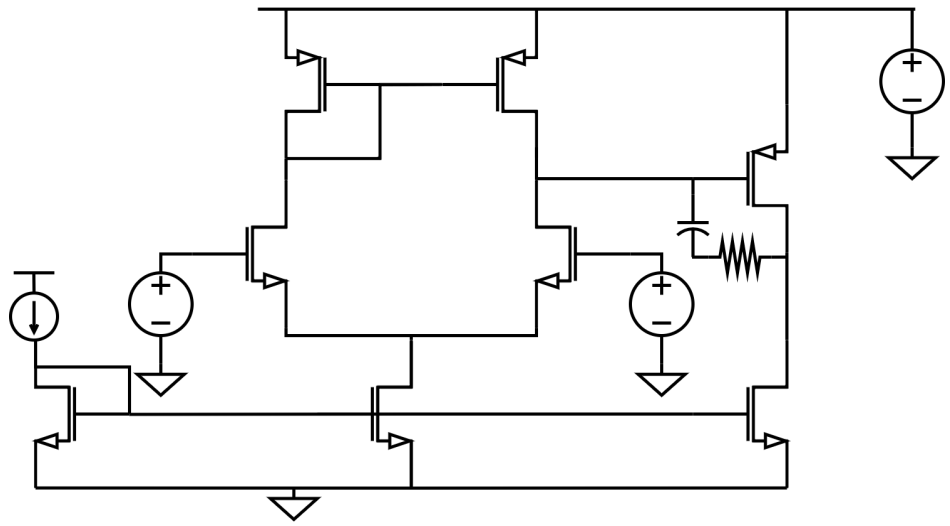


Figure 5.5: The circuit schematic of the pretraining dataset for the OpAmp experiments.

common-mode input, supply input, .etc). To get the ground truth we run Spice simulation on each instance and store all DC voltages along with the graph representation of the circuit. During test time, we evaluate our predictions on new topological modifications of this circuit where the current mirror self-bias connection is converted to a voltage-biased connection (Figure 5.8a). This minor topological modification induces enough behavioral change to the circuit that for proper generalization, the model has to have learned fine details on device-device interaction during pre-training.

Neural network architecture design

For all experiments, we opted to use the DeepGEN architecture proposed in [68] as the GNN backbone. This architecture has been proposed as a solution to train very deep Graph Convolution Networks. It leverages generalized learnable aggregation layers as well as pre-activation skip connections to battle the vanishing gradient and over-smoothing problem often seen in making GNNs deep. We found that this architecture generally outperforms other vanilla GNN architectures such as GCN [59], GAT [124].

Table 5.1: Test Acc@100 of prediction of voltages for each class of resistor ladders. We ran each experiment 3 times with random seeds for getting the standard errors.

	R12	R16	R32	R128
MLP-5-fixed	0.3064 ± 0.0079	0.2466 ± 0.0056	0.1649 ± 0.0014	0.0905 ± 0.0009
DeepGEN-5-fixed	0.1024 ± 0.0073	0.1073 ± 0.0071	0.1114 ± 0.0084	0.1150 ± 0.0038
DeepGEN-10-fixed	0.6984 ± 0.0090	0.6585 ± 0.0110	0.6142 ± 0.0392	0.6167 ± 0.0027
DeepGEN-15-fixed	0.9124 ± 0.0112	0.8756 ± 0.0493	0.8756 ± 0.0166	0.8329 ± 0.0419
DeepGEN-5-(FPT)	0.1029 ± 0.0013	0.1038 ± 0.0026	0.1001 ± 0.0027	0.1009 ± 0.0027
DeepGEN-10-(FPT)	0.6847 ± 0.0069	0.6493 ± 0.0039	0.5979 ± 0.0064	0.5660 ± 0.0095
DeepGEN-15-(FPT)	0.8980 ± 0.0068	0.8711 ± 0.0059	0.8406 ± 0.0127	0.8186 ± 0.0093

Do we learn to model DC voltages?

We pre-train the NN parameters with an Adam optimizer with a batch size of 256 until validation accuracy reaches its maximum. Figure 5.6 shows the train and validation Acc@K for both resistor-ladder (top) and opamp (bottom) circuits. The validation set is simply a random split on the original dataset and it is clear that we can successfully reach $> 90\%$ accuracy for both problems during pretraining.

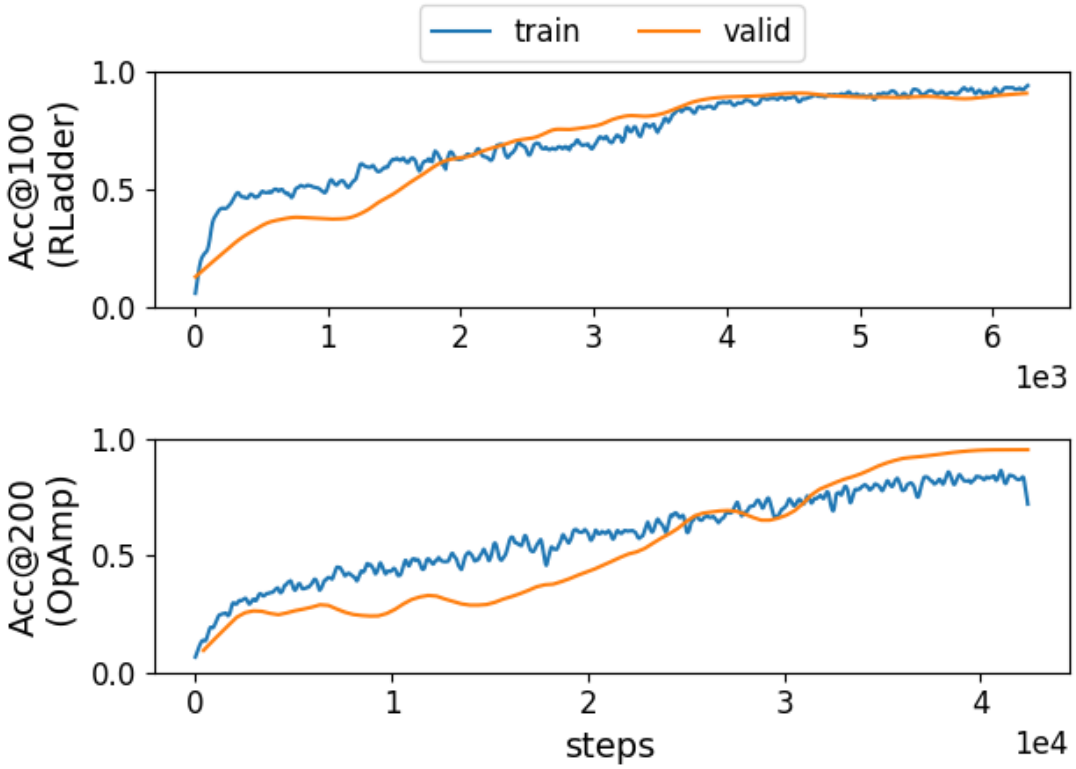


Figure 5.6: Train and Valid Accuracy of the model during pretraining on the resistor ladder circuit (top) and OpAmp circuit (bottom). Validation Acc@100 for resistor ladder reaches to 92% and Acc@200 for OpAmp reaches to 91% during pretraining.

Generalization to new unseen scenarios

Zero-shot generalization to new topologies

We first investigate whether the GNN backbone and the proceeding MLP (architecture of Figure 5.2) can generalize their prediction to new unseen topologies of resistor ladders with more branches and hence more complexity. To this end, we generate separate graph datasets for resistor ladders with 12, 16, 32, and 128 branches. A single graph example with 10 branches (the largest graph in the pretraining dataset) has 84 nodes, and 154 edges while with 128 branches it has 1032 nodes, and 2310 edges, an order of magnitude increase in graph complexity. For each circuit type, we generate 1k circuit instances as test set and 20k other instances for training other baselines for comparison. We consider the following approaches as baselines:

1. We take the pre-trained model, freeze it and evaluate its prediction capability in a zero-shot manner with no finetuning. We refer to this approach as **FPT** (frozen pre-trained)

model. In this modeling scheme, there is a topological mismatch between train and test sets. The model only gets trained on graphs with 2 to 10 branches but is then evaluated on 12, 16, 32, and 128 branches which are more complex and require domain knowledge of how resistors and voltage sources behave in the context of resistor ladders.

2. We use the 20k downstream training set to train a *separate* predictive model, specialized for each branch size. In this modeling scheme, there is no mismatch between train and test sets. Note that the size of this dataset is similar to the size of the pre-training dataset. For this strategy, since the input has a fixed static shape we can also represent it as a fixed size vector by concatenating the node features into a vector and using an MLP backbone as the node feature extractor. This is referred to as **MLP-5-fixed** since it has 5 layers. By comparing this model and GNNs we can see the impact of including the structural information in the model architecture. The specialized GNN models are referred to as **DeepGEN-x-fixed** in Table 5.1 (where x is the number of layers).

Results. Table 5.1 compares the Acc@100 of the predictions of GNN backbones with different number of layers. All the models were trained with the same MSE loss, with a batch size of 256 using Adam optimizer[2]. We first notice that even though FPT models were never trained on the test topologies, they have an on par accuracy with the specialized models that were separately trained on comparable dataset size from a single topology.

Moreover, we also note that deeper models have better generalization across the board. This could be due to the lack of a large enough receptive field over the entire graph for capturing long-range dependencies in shallower models.

Also, as evident by the poor performance of the results of MLP-5-fixed in Table 5.1, we conclude that the structural information built into the model architecture of GNNs can significantly improve the generalization of the model. This is important evidence for choosing GNNs over MLPs even in a fixed topology setup since they have a better inductive bias for modeling circuits.

Few-shot generalization to other unseen topologies

In this experiment, we evaluate the FPT models on circuits that are slightly more different than the circuits that they have been pre-trained on (e.g. resistor divider with parallel or series resistor combinations in case of the resistor ladder pre-training dataset). These small modifications, which are highlighted with dashed red lines in Figure 5.7 and 5.8a, will result in poor zero-shot performance, but we show that we can significantly improve the prediction performance via fine-tuning on small datasets from these topologies. For resistor-based circuits, we generate 1k graphs per each topology for training and then a separate 1k instance held-out dataset for evaluation. For the OpAmp we generate 10k samples for training and 1k examples for evaluation from the topology shown in Figure 5.8a in which the self-biased current mirror connection is removed and the gate connections are replaced by a voltage source. For baselines, we compare the finetuned models (FT-PT) against training the

[2]Only for DeepGEN-15-fixed the batch size had to be reduced from default of 256 to 32 and 16 for R32 and R128, respectively just to prevent exceeding the GPU memory

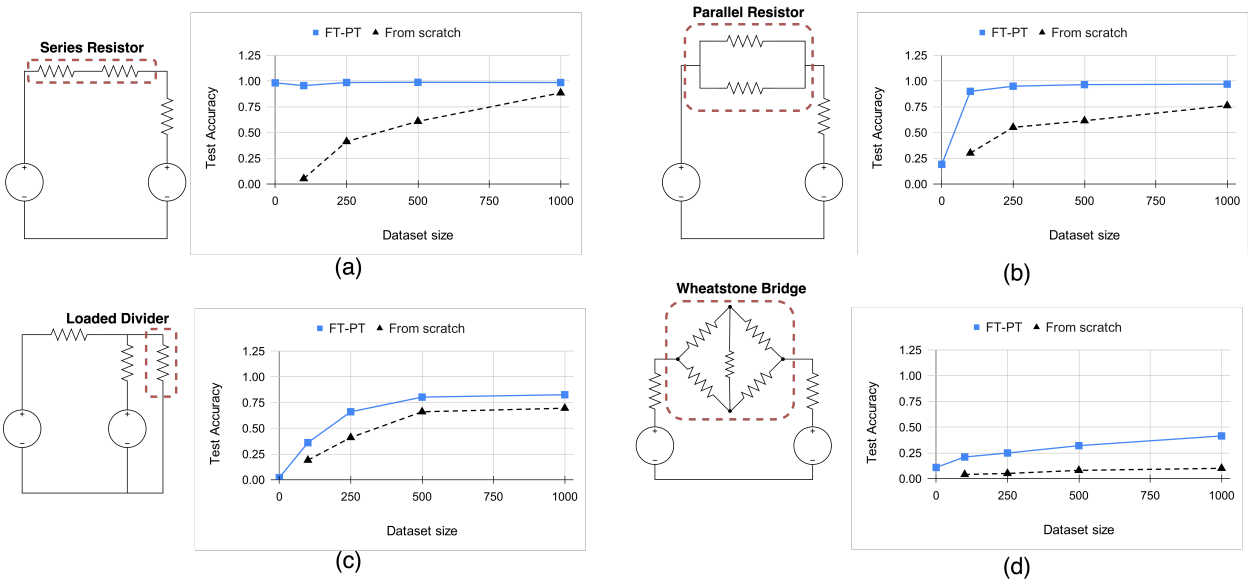


Figure 5.7: Fine-tuning capabilities of the pretrained model to new unseen topologies comprised of resistors and voltages sources. Each figure shows models that are trained and evaluated on the illustrated resistor networks where new unseen topological configurations are highlighted in red dashed boxes.

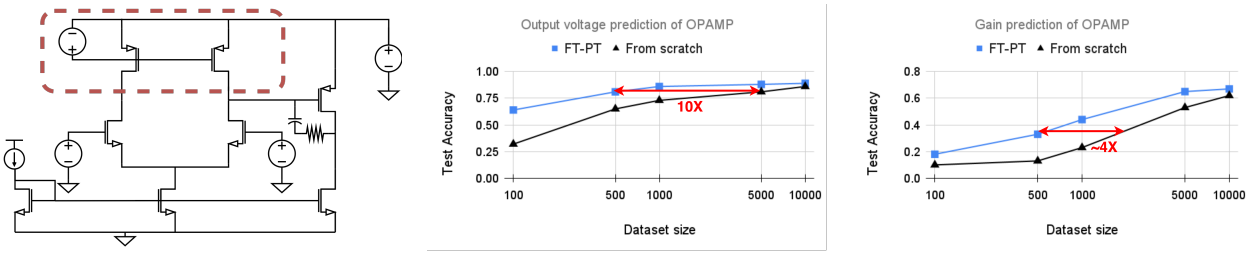
same architecture from scratch. For the remainder of the experiments, we use DeepGEN-15 as the GNN backbone unless otherwise mentioned.

Results. Figures 5.7 and 5.8b show this comparison for each circuit. For resistive circuits, we finetune the backbone on 10, 25, 50, and 100% of the training dataset and for the OpAmp we finetune with 1, 5, 10, 50, and 100 % of the dataset. We can see that even with a severe distribution shift from pretraining and test domains, finetuning the learned representation can be more efficient than learning from scratch.

Domain knowledge transfer by re-using the learned GNN features for predicting graph level tasks

In this section, we use the architecture outlined in Figure 5.2 to re-use the learned node features of the pre-trained GNN by using a cross attention pooling layer to construct a graph feature from aggregating node features and use that for prediction of graph-level metrics. To learn the model, we update all parameters (including the GNN backbone parameters) using Adam optimizer on the downstream training dataset.

For this experiment, we have generated 10k instances of circuits from Figure 5.8a that have ground-truth values of the gain of the amplifiers which are collected by running simulations. We then evaluate the generalization of our learned models (trained by fine-tuning



(a) (b) (c)

Figure 5.8: (a) The unseen topology of the two-stage opamp that the GNN gets fine-tuned on. We collect a small dataset of 10 thousand designs from this topology along with their ground truth output values. We then use fractions of this training dataset to show how much improvements we get from more data. (b) shows the test Acc@200 of predicting the output voltage for our fine-tuned method (FT-PT) vs. training from scratch with no knowledge transfer. (c) shows the test Acc@50 comparison for the gain prediction task (i.e. also a new graph property prediction task).

or from scratch) on 1k separate held-out data points collected in the same procedure.

Results. Figure 5.8c shows the accuracy of gain prediction after fine-tuning on various dataset sizes. As illustrated in this figure we can see that the pre-trained features can significantly improve the generalization capabilities of the model even with small fractions of the training data.

To study how important the choice of using an attention-based pooling layer is over simpler alternative pooling methods, we have also conducted an ablation experiment where we compare our method against a simple average pooling of all node features to construct the graph feature. We perform this ablation, on the task of predicting the output resistance of the resistor ladders with 10 branches. Figure 5.9 compares these baselines. As seen in the figure, our method outperforms the simple average pooling method usually used in literature [51]. We can also see that pre-training the GNNs prevent the model from overfitting to the small available downstream dataset as it does if trained from scratch.

Improving sample efficiency of model-based optimization by using pre-trained models

In this section, we show the benefits of pre-trained circuit models in improving the sample efficiency of model-based circuit optimization algorithms like BagNet. In these methods, part of the objective is to learn a good model of the design objectives so that we can compare different design choices and focus exploration on more promising regions.

In chapter 4, a simple multi-layer perceptron was used as the backbone feature extractor which was shown to out-perform the base evolutionary algorithm with no such discriminator.

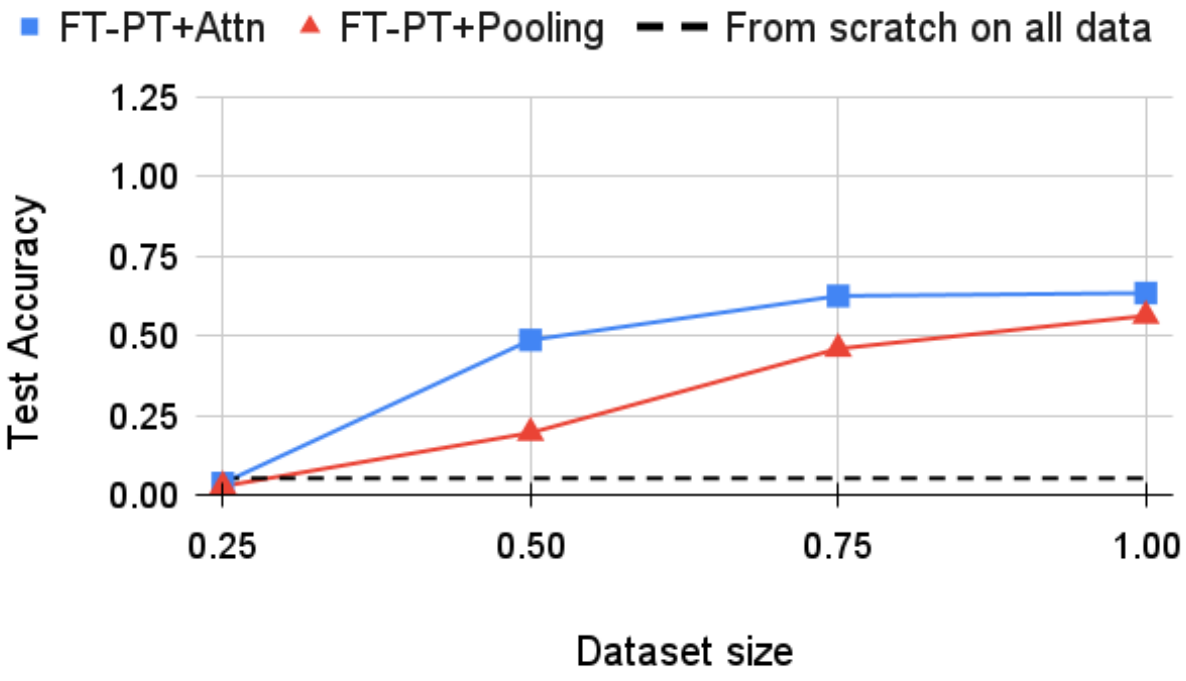


Figure 5.9: Acc@100 of the output resistance prediction task on resistor ladder. Training from scratch will fail to generalize to the test set even if all 1k data points are used due to significant over-fitting. Moreover the attention-based aggregation layer outperforms the naive average pooling aggregation.

In this section, we reproduce those results and show that if we use the pre-trained GNN backbone for extracting circuit features we can significantly boost the performance to almost as good as an oracle discriminator which has access to the ground truth design metrics and their comparisons. To validate our hypothesis, we use the Operational Amplifier introduced in Figure 5.5 as our pretraining topology. In this circuit, the compensation method is comprised of a series capacitor and resistor ($C_c - R_z$). However, for optimization we consider two target topologies: one is the same as the pre-training topology and another one with a different topology where only a capacitor is used for compensation (C_c). It is known that adding a resistor can make the compensation easier to expand the bandwidth of the amplifier. The design specification requirements are similar to those presented in Table 4.1.

We consider the following baselines:

- **Evolutionary (Evo)** just runs the base evolutionary algorithm and does not use any discriminator during the optimization
- **Oracle** uses ground truth comparison labels for the discriminator

Table 5.2: The comparison of sample complexity of the optimization algorithms.

	$C_c - R_z$		$C_c - only$	
	Queries	Sims	Queries	Sims
Oracle	3770.7	193.0	18821.0	224.3
Evo	-	2757.3	-	5422.7
BagNet + FC	14720.0	397.7	176568.3	4024.0
BagNet + Randinit GNN	8892.0	254.7	47961.7	426.5
BagNet + FPT GNN	8000.3	347.3	27309.0	500.3
BagNet + FT-PT GNN (Ours)	**4890.7**	**226.0**	**14408.7**	**293.0**

- **BagNet + FC** uses fully-connected layers as feature extractor (the original BagNet work)
- **BagNet + Randinit GNN**, uses a randomly initialized GNN as feature extractor that is jointly trained with the the rest of the model during optimization
- **BagNet + FPT GNN** uses a frozen but pre-trained GNN for feature extractor
- **BagNet + FT-PT GNN** fine-tunes the pretrained GNN

Results. Figure 5.10 qualitatively shows the quality of the designs found during optimization as the function of the number of iterations. We run each algorithm with three initial seeds to show the standard error on performance. The y-axis measures the objective value of the top 20 designs found until each iteration. We also provide the average number of simulations and the average number of discriminator queries for all of these baselines in Table 5.2 for quantitative comparisons.

All BagNet algorithms that use the discriminator outperform the base evolutionary algorithm. We note that the performance boost obtained by the pre-trained and fine-tuned GNN (FT-PT) is very close to the oracle baseline while using the frozen pre-trained features or randomly initialized GNN is not as good. This gap is larger when tried on the harder optimization problem (C_c-only), despite pretraining FT-PT's initial weights on a different topology than the target optimization topology. This picture clearly shows that the performance boost in sample efficiency is partly due to the utilization of GNNs over MLP architectures as feature extractors and partly due to being able to transfer the pre-trained representations to unseen topologies. We also note that the number of discriminator queries is typically lower when the model is better. This implies that the more accurate the model gets the number of new candidate designs that get rejected by mistake become lower.

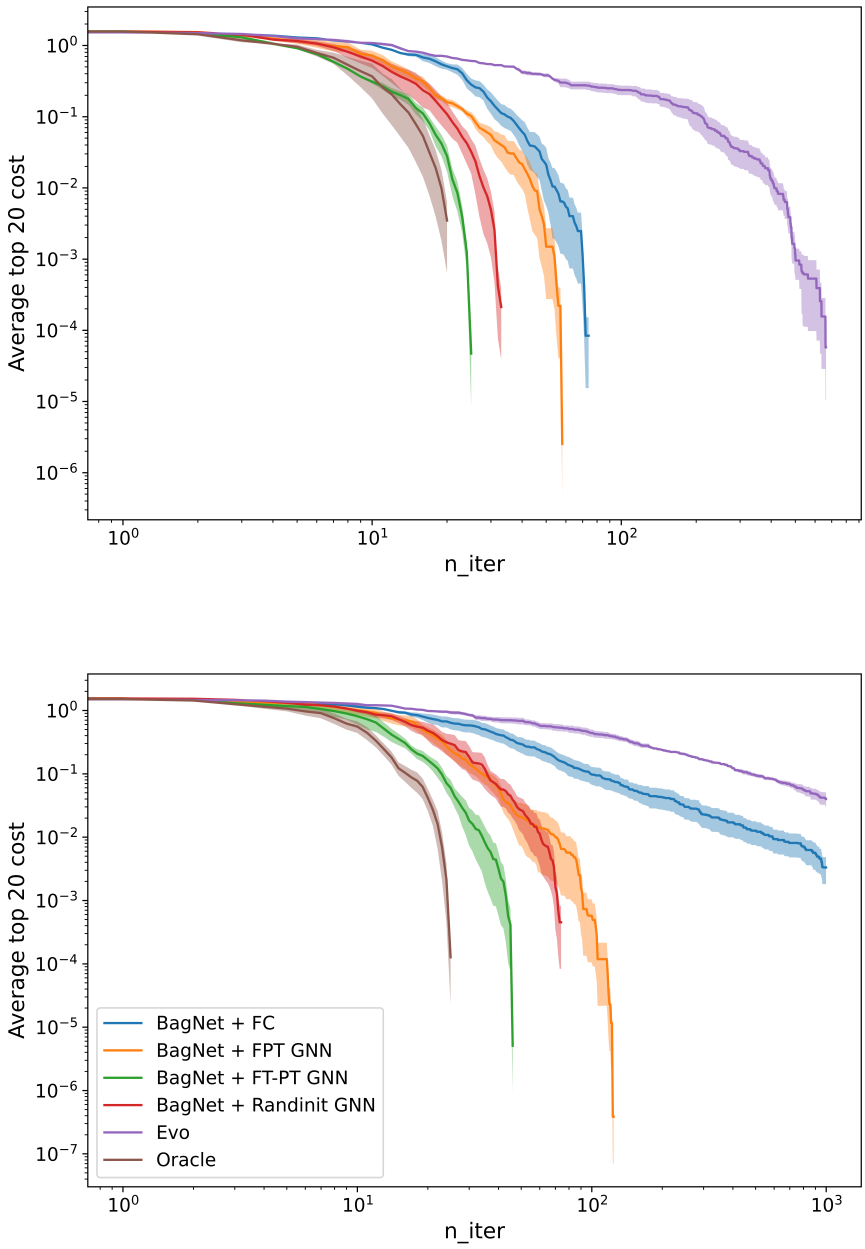


Figure 5.10: The average cost of top 20 designs over the course of optimization using different evolutionary-based approaches. (Top) The optimization is done on the OpAmp with $C_c - R_z$ compensation (Fig. 5.5, (Bottom) The target topology is the OpAmp with only capacitor compensation (C_c)

5.4 Conclusion

This work illustrates how we can use large in-expensive datasets collected from running DC simulations at scale on analog circuits to learn transferable circuit representations that could enable more sample efficient modeling and optimization of new expensive-to-collect metrics.

We presented a supervised-learning paradigm for learning representations that are transferable to new circuit topologies that can be fine-tuned for new graph-level prediction tasks. We introduced a novel graph representation of the circuit and used the prediction of the output node voltages as the pretraining objective. We showed that such a pretraining objective can induce learning generalizable domain-specific features that can be easily fine-tuned for similar predictions on new topologies or unseen new graph property prediction tasks such as predicting the output resistance of a circuit or gain of an amplifier. We have also shown that we can leverage the pre-trained features to boost the performance of model-based optimization methods used for analog circuit design automation.

Chapter 6

Boosting Sample efficiency of Bayesian Optimization Methods via Pre-training

Bayesian optimization (BO) is a popular framework for expensive-to-evaluate black-box optimization problems as it seeks to minimize the number of function evaluations required for optimizing a target black-box function [111, 28]. In real-world scenarios, however, we often have access to offline evaluations of one or more auxiliary black-box functions related to the target function. For example, in the circuit design domain, designers are interested in performing *layout* simulations for satisfying the required design configurations. These simulations are however expensive to run; designers instead often turn to *schematic* simulations as an inexpensive proxy for prototyping designs. Multi-task Bayesian optimization (MBO) is an optimization paradigm that extends BO to exploit such additional sources of information from related black-box functions for efficient optimization [120].

When offline datasets are large, the scalability of prior approaches comes at the expense of expressivity and inference quality. In this chapter, we propose JUMBO (Joint Upper Confidence Bound Multi-task Bayesian Optimization), an MBO algorithm that sidesteps these limitations by querying additional data based on a combination of acquisition signals derived from training two Gaussian Processes (GP): a *cold-GP* operating directly in the input domain and a *warm-GP* that operates in the feature space of a deep neural network pre-trained using the offline data. Such a decomposition can dynamically control the reliability of information derived from the online and offline data and the use of pre-trained neural networks permits scalability to large offline datasets.

6.1 Background

We are interested in maximizing a target black-box function $f : \chi \to \mathbb{R}$ defined over a discrete or compact set $\chi \subseteq \mathbb{R}^d$. We assume only query access to f. For every query point x,

we receive a noisy observation $y = f(x) + \epsilon$. Here, we assume ϵ is standard Gaussian noise, i.e., $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ where σ_n is the noise standard deviation. Our strategy for optimizing f will be to learn a probabilistic model for regressing the inputs x to y using the available data and using that model to guide the acquisition of additional data for updating the model. In particular, we will be interested in using Gaussian Process regression models within a Bayesian Optimization framework, as described below.

Gaussian Process (GP) Regression

A Gaussian Process (GP) is defined as a set of random variables such that any finite subset of them follows a multivariate normal distribution. A GP can be used to define a prior distribution over the unknown function f, which can be converted to a posterior distribution once we observe additional data. Formally, a GP prior is defined by a mean function $\mu_0 : \mathcal{X} \to \mathbb{R}$ and a valid kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. A kernel function κ is valid if it is symmetric and the Gram matrix K is positive semi-definite. Intuitively, the entries of the kernel matrix $K_{i,j} = \kappa(x_i, x_j)$ measure the similarity between any two points x_i and x_j. Given points $X = \{x_1, x_2, \ldots, x_n\}$, the distribution of the function evaluations $\mathbf{f} = [f(x_1), f(x_2), \ldots f(x_n)]$ in a GP prior follows a normal distribution, such that $\mathbf{f}|X \sim \mathcal{N}(\mu_0(X), K(X, X))$ where $\mu_0(X) = [\mu_0(x_1), \mu_0(x_2), \ldots \mu_0(x_n)]$ and $K(X, X)$ is a covariance matrix. For simplicity, we will henceforth assume μ_0 to be a zero mean function.

Given a training dataset $\mathcal{D}$, let $X_\mathcal{D}$ and $\mathbf{y}_\mathcal{D}$ denote the inputs and their noisy observations. Since the observation model is also assumed to be Gaussian, the posterior over f at a test set of points X^* will follow a multivariate normal distribution with the following mean and covariance:

$$\begin{aligned}
\mu(\mathbf{f}^*|\mathcal{D}, X^*) &= K(X^*, X_\mathcal{D})^T \tilde{K}_D^{-1} \mathbf{y}_\mathcal{D}, \\
\Sigma(\mathbf{f}^*|\mathcal{D}, X^*) &= K(X^*, X^*) - K(X^*, X_\mathcal{D})^T \tilde{K}_D^{-1} K(X^*, X_\mathcal{D}), \\
\text{where } \tilde{K}_D &= K(X_\mathcal{D}, X_\mathcal{D}) + \sigma_n^2 I.
\end{aligned}$$

Due to the inverse operation during posterior computation, standard GPs can be computationally prohibitive for modeling large datasets.

We direct the reader to [103] for an overview on GPs.

Bayesian Optimization (BO)

Bayesian Optimization (BO) is a class of sequential algorithms for sample-efficient optimization of expensive black-box functions [28, 111]. A BO algorithm typically runs for a fixed number of rounds. At every round t, the algorithm selects a query point x_t and observes a noisy function value y_t. To select x_t, the algorithm first infers the posterior distribution over functions $p(\mathbf{f}|\{(x_i, y_i)\}_{i=1}^{t-1})$ via a probabilistic model (e.g., Gaussian Processes). Thereafter, x_t is chosen to optimize an uncertainty-aware acquisition function that balances exploration and exploitation. For example, a popular acquisition function is the Upper Confidence Bound

(UCB) which prefers points that have high expected value (exploitation) and high uncertainty (exploration). With the new point (x_t, y_t), the posterior distribution can be updated and the whole process is repeated in the next round.

At round t, we define the instantaneous regret as $r_t = f(x^*) - f(x_t)$ where x^* is the global optima and x_t maximizes the acquisition function. Similarly, we can define the cumulative regret at round T as the sum of instantaneous regrets $R_T = \sum_{t=1}^{T} r_t$. A desired property of any BO algorithms is to be *no-regret* where the cumulative regret is sub-linear in T as $T \to \infty$, i.e., $\lim_{T\to\infty} {}^{R_T}/_T = 0$.

Multi-task Bayesian Optimization (MBO)

Our focus setting in this work is a variant of BO, called Multi-Task Bayesian Optimization (MBO). Here, we assume $K > 0$ auxiliary real-valued black-box functions $\{f_1, \ldots, f_K\}$, each having the same domain χ as the target function f [120, 116]. For each function f_k, we have an offline dataset $\mathcal{D}^{(k)}$ consisting of pairs of input points x and the corresponding function evaluations $f_k(x)$. If these auxiliary functions are related to the target function, then we can transfer knowledge from the offline data $\mathcal{D}^{\text{aux}} = \mathcal{D}^{(1)} \cup \cdots \cup \mathcal{D}^{(K)}$ to improve the sample-efficiency for optimizing f. In certain applications, we might also have access to offline data from f itself. However, in practice, f is typically expensive to query and its offline dataset $\mathcal{D}^f$ will be very small.

We discuss some prominent works in MBO that are most closely related to our proposed approach below. See Section 6.2 for further discussion about other relevant work.

Multi-task BO [120] is an early approach that employs a custom kernel within a multi-task GP [130] to model the relationship between the auxiliary and target functions. Similar to standard GPs, multi-task GPs fail to scale for large offline datasets.

On the other hand, parametric models such as neural networks (NN), can effectively scale to larger datasets but do not defacto quantify uncertainty. Hybrid methods such as **DNGO** [115] achieve scalability for (single task) BO through the use of a feed-forward deep NN followed by Bayesian Linear Regression (BLR) [7]. The NN is trained on the existing data via a simple regression loss (e.g, mean squared error). Once trained, the NN parameters are frozen and the output layer is replaced by BLR for the BO routine. For BLR, the computational complexity of posterior updates scales linearly with the size of the dataset.

This step can be understood as applying a GP to the output features of the NN with a linear kernel (i.e. $\kappa(x_i, x_j) = h_\phi(x_i)^T h_\phi(x_j)$ where h is the NN function with parameters ϕ).

For BLR, the computational complexity of posterior inference is linear w.r.t. the number of data points and thus DNGO can scale to large offline datasets.

MT-ABLR [89] extends DNGO to multi task settings by training a single NN to learn a shared representation $h_\phi(x)$ followed by task-specific BLR layers (i.e. predicting $f_1(x), ..., f_K(x)$, and $f(x)$ based on inputs). The learning objective corresponds to the maximization of sum of the marginal log-likelihoods for each task: $\sum_{t=1}^{K+1} p(\mathbf{y}_t | w_t, h_\phi(X_t), \sigma_t)$. The main task is included in the last index, w_t is the Bayesian Linear layer weights for task

t with prior $p(w_t) = \mathcal{N}(0, \sigma_{w_t}^2 I)$, σ_t and σ_{w_t} are the hyper-prior parameters, and $(X_t, \mathbf{y}_t)$ is the observed data from task t. Learning $h_\phi(x)$ by directly maximizing the marginal likelihood improves the performance of DNGO while maintaining the computational scalability of its posterior inference in case of large offline data. However, both DNGO and ABLR have implicit assumptions about the existence of a feature space under which the target function can be expressed as a linear combination. This can be a restrictive assumption and there is no guarantee that given finite data such feature space can be learned.

MT-BOHAMIANN [116] addresses the limited expressivity of prior approaches by employing Bayesian NNs to specify the posterior over $\mathbf{f}$ and feed the NN with input x and additional learned task-specific embeddings $\psi(t)$ for task t.

While allowing for a principled treatment of uncertainties, fully Bayesian NNs are computationally expensive to train and their performance depends on the approximation quality of stochastic gradient HMC methods used for posterior inference.

6.2 Related Work

Transfer Learning in Bayesian Optimization: Utilizing prior information for applying transfer learning to improve Bayesian optimization has been explored in several prior papers. Early work of [120] focuses on the design of multi-task kernels for modeling task correlations [93]. These models tend to suffer from a lack of scalability; [132, 24] show that this challenge can be partially mitigated by training an ensemble of task-specific GPs that scale linearly with the number of tasks but still suffer from cubic complexity in the number of observations for each task. To address scalability and robust treatment of uncertainty, several prior works have been suggested [107, 116, 89]. [107] employs a Gaussian Copula to learn a joint prior on hyper-parameters based on offline tasks, and then utilizes a GP on the online task to adapt to the target function. [116] uses a BNN as surrogates for MBO; however, since training BNNs is computationally intensive [89] proposes to use a deterministic NN followed by a BLR layer at the output to achieve scalability.

Some other prior work exploits certain assumptions between the source and target data. For example, [113, 34] assume an ordering of the tasks and use this information to train GPs to model residuals between the target and auxiliary tasks. [25, 131] assume the existence of a similarity measure between prior and target data which may not be easy to define for problems other than hyper-parameter optimization. A simpler idea is to use prior data to confine the search space to promising regions [88]. However, this highly relies on whether the confined region includes the optimal solution to the target task. Another line of work studies utilizing prior optimization runs to meta-learn acquisition functions [127]. This idea can be utilized in addition to our method and is not a competing direction.

Multi-fidelity Black-box Optimization (MFBO): In multi-fidelity scenarios, we can query for noisy approximations to the target function relatively cheaply. For example, in hyperparameter optimization, we can query for cheap proxies to the performance of a configuration on a smaller subset of the training data [91], early stopping [69], or by predicting

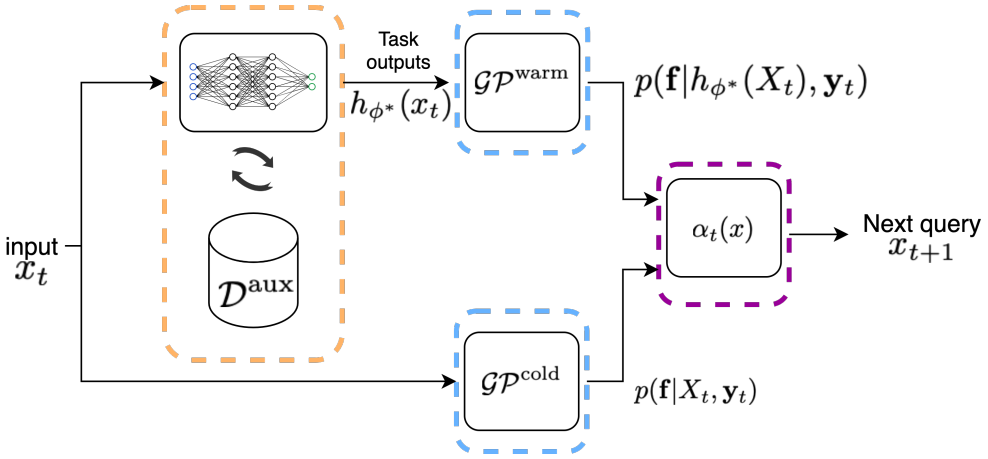


Figure 6.1: JUMBO. During the pretraining phase, we learn a NN mapping h_{ϕ^*} (orange) for the warm-GP. The next query based on α_t (purple) will be the point that has a high score based on the acquisition function of both warm and cold GP (blue).

learning curves [19, 61]. We direct the reader to Section 1.4 in [53] for a comprehensive survey on MFBO. Such methods, similar to MF-GP-UCB [58] (section 6.5), are typically constrained to scenarios where such low fidelities are explicitly available and make strong continuity assumptions between the low fidelities and the target function.

Deep Kernel Learning (DKL): Commonly used GP kernels (e.g. RBF, Matern) can only capture simple correlations between points a priori. DKL [52, 11] addresses this issue by learning a latent representation via NN that can be fed to a standard kernel at the output. [115] employs linear kernels at the output of a pre-trained NN while [52] extends it to use non-linear kernels. The *warm-GP* in JUMBO can be understood as a DKL surrogate model trained using offline data from auxiliary tasks.

6.3 Scalable MBO via JUMBO

In the previous section, we observed that prior MBO algorithms make trade-offs in either posterior expressivity or inference quality to scale to large offline datasets. Our goal is to show that these trade-offs can be significantly mitigated and consequently, the design of our proposed MBO framework, which we refer to as **J**oint **U**pper confidence **M**ulti-task **B**ayesian **O**ptimization (JUMBO), will be guided by the following desiderata: (1) Scalability to large offline datasets (e.g., via NNs) (2) Exact and computationally tractable posterior updates (e.g., via GPs) (3) Flexible and expressive posteriors (e.g., via non-linear kernels).

Regression Model

The regression model in JUMBO is composed of two GPs: a *warm-GP* and a *cold-GP* denoted by $\mathcal{GP}^{\text{warm}}(0, \kappa^w)$ and $\mathcal{GP}^{\text{cold}}(0, \kappa^c)$, respectively. As shown in Figure 6.1, both GPs are trained to model the target function f but operate in different input spaces, as we describe next.

$\mathcal{GP}^{\text{warm}}$ (with hyperparameters θ_w) operates on a feature representation of the input space $h_\phi(x)$ derived from the offline dataset $\mathcal{D}^{\text{aux}}$. To learn this feature space, we train a multi-headed feed-forward NN to minimize the mean squared error for each auxiliary task, akin to DNGO [115]. Thereafter, in contrast to both DNGO and ABLR, we do not train separate output BLR modules. Rather, we will directly train $\mathcal{GP}^{\text{warm}}$ on the output of the NN using the data acquired from the target function f. Note that for training $\mathcal{GP}^{\text{warm}}$, we can use any non-linear kernel, which results in an expressive posterior that allows for exact and tractable inference using closed-form expressions.

Additionally, we can encounter scenarios where some of the auxiliary functions are insufficient in reducing the uncertainty in inferring the target function. In such scenarios, relying solely on $\mathcal{GP}^{\text{warm}}$ can significantly hurt performance. Therefore, we additionally initialize $\mathcal{GP}^{\text{cold}}$ (with hyperparameters θ_c) directly on the input space χ.

If we also have access to offline data from f (i.e. $\mathcal{D}^f$), the hyperparameters of the warm and cold GPs can also be pre-trained jointly along with the neural network parameters. The overall pre-training objective is then given by:

$$\mathcal{L}(\phi, \theta_w, \theta_c) = \mathcal{L}^{\text{MSE}}(\phi|\mathcal{D}^{\text{aux}}) + \mathcal{L}^{\mathcal{GP}}(\theta_w|\mathcal{D}^f) + \mathcal{L}^{\mathcal{GP}}(\theta_c|\mathcal{D}^f) \tag{6.1}$$

where $\mathcal{L}^{\mathcal{GP}}(\cdot|\mathcal{D}^f)$ denotes the negative marginal log-likelihood for the corresponding GP on $\mathcal{D}^f$.

Acquisition Procedure

Post the offline pre-training of the JUMBO's regression model, we can use it for online data acquisition in a standard BO loop. The key design choice here is the acquisition function, which we describe next. At round t, let $\alpha_t^{\text{warm}}(x)$ and $\alpha_t^{\text{cold}}(x)$ be the single task acquisition function (e.g. UCB) of the warm and cold GPs, after observing $t-1$ data points, respectively.

Our guiding intuition for the acquisition function in JUMBO is that we are most interested in querying points that are scored highly by both acquisition functions. Ideally, we want to first sort points based on α^{warm} scores and then from the top choices select the ones with the highest α^{cold} score.

To realize this acquisition function on a continuous input domain, we define it as a convex combination of the individual acquisition functions by employing a *dynamic* interpolation coefficient $\lambda_t(x) \in [0, 1]$. Formally,

$$\alpha_t(x) = \lambda_t(x)\alpha_t^{\text{cold}}(x) + (1 - \lambda_t(x))\alpha_t^{\text{warm}}(x). \tag{6.2}$$

In Eq. 6.2, By choosing $\lambda_t(x)$ to be close to 1 for points with $\alpha_t^{\text{warm}}(x) \approx \max_x \alpha_t^{\text{warm}}(x)$, we can ensure to acquire points that have high acquisition scores as per both $\alpha_t^{\text{cold}}(x)$ and $\alpha_t^{\text{warm}}(x)$. Next, we will discuss some theoretical results that shed more light on the design of $\lambda_t(x)$.

6.4 Theoretical Analysis

Next, we will formally derive the regret bound for JUMBO and provide insights on the conditions under which JUMBO outperforms GP-UCB [118]. For this analysis, we will use Upper Confidence Bound (UCB) as our acquisition function for the warm and cold GPs. To do so, we utilize the notion of Maximum Information Gain (MIG).

Definition 1 (Maximum Information Gain [118])**.** *Let $f \sim \mathcal{GP}(0, \kappa)$, $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. Consider any $\mathcal{X} \subset \mathbb{R}^d$ and let $\tilde{\mathcal{X}} = \{x_1, ..., x_n\} \subset \mathcal{X}$ be a finite subset. Let $\mathbf{y}_{\tilde{\mathcal{X}}} \in \mathbb{R}^n$ be n noisy observations such that $(\mathbf{y}_{\tilde{\mathcal{X}}})_i = (\mathbf{f}_{\tilde{\mathcal{X}}})_i + \epsilon_i$, $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$. Let I denote the Shannon mutual information.*

The MIG $\Psi_n(\mathcal{X})$ of set $\mathcal{X}$ after n evaluations is the maximum mutual information between the function values and observations among all choices of n points in $\mathcal{X}$. Formally,

$$\Psi_n(\mathcal{X}) = \max_{\tilde{\mathcal{X}} \subset \mathcal{X}, |\tilde{\mathcal{X}}| = n} I(\mathbf{y}_{\tilde{\mathcal{X}}}; \mathbf{f}_{\tilde{\mathcal{X}}})$$

This quantity depends on kernel parameters and the set $\mathcal{X}$, and also serves as an important tool for characterizing the difficulty of a GP-bandit. For a given kernel, it can be shown that $\Psi_n(\mathcal{X}) \propto \Pi(\mathcal{X})$ where $\Pi(\mathcal{X}) = |\mathcal{X}|$ for discrete and $\text{Vol}(\mathcal{X})$ for the continuous case [118]. For example for *Radial Basis* kernel $\Psi_n([0,1]^d) \in O(\log(n)^{d+1})$. We will first focus on settings where $\mathcal{X}$ is discrete.

For GP-UCB [118], it has been shown that for any $\delta \in (0, 1)$, if $f \sim \mathcal{GP}(0, \kappa)$ (i.e., the GP assigns non-zero probability to the target function f), then the cumulative regret R_T after T rounds will be bounded with probability at least $1 - \delta$:

$$Pr\{R_T \leq \sqrt{CT\beta_T \Psi_T(\mathcal{X})}, \forall T \geq 1\} \geq 1 - \delta \tag{6.3}$$

with $C = \frac{8}{\log(1+\sigma_n^{-2})}$ and $\beta_T = 2\log\left(\frac{|\mathcal{X}|\pi^2 T^2}{6\delta}\right)$.

Recall that $h_\phi : \mathcal{X} \to \mathcal{Z}$ is a mapping from input space $\mathcal{X}$ to the feature space $\mathcal{Z}$. We will further make the following modeling assumptions to ensure that the target black-box function f is a sample from both the cold and warm GPs.

Assumption 1. $f \sim \mathcal{GP}^{cold}(0, \kappa^c)$.

Assumption 2. *Let ϕ^* denote the NN parameters obtained via pretraining (Eq. 7.3).Then, there exists a function $g \sim \mathcal{GP}^{warm}(0, \kappa^w)$ such that $f = g \circ h_{\phi^*}$.*

Theorem 1. *Let $\mathcal{X}_g \subset \mathcal{X}$ and $\bar{\mathcal{X}}_g = \mathcal{X} \setminus \mathcal{X}_g$ be some arbitrary partitioning of the input domain $\mathcal{X}$. Define the interpolation coefficient as an indicator $\lambda_t(x) = \mathbb{1}(x \in \mathcal{X}_g)$. Then under Assumptions 1 and 2, JUMBO is no-regret.*

Specifically, let s be the number of rounds such that the JUMBO queries for points $x_t \in \bar{\mathcal{X}}_g$. Then, for any $\delta \in (0,1)$, running JUMBO for T iterations results in a sequence of candidates $(x_t)_{t=1}^{t=T}$ for which the following holds with probability at least $1-\delta$:

$$R_T < \sqrt{CT\beta_T\{\Psi_{T-s}(\mathcal{X}_g) + \Psi_s(\bar{\mathcal{Z}}_g)\}}, \forall T \geq 1 \tag{6.4}$$

where $C = \frac{8}{\log\left(1+\sigma_n^{-2}\right)}$, $\beta_t = 2\log\left(\frac{|\mathcal{X}|\pi^2 t^2}{3\delta}\right)$, and $\bar{\mathcal{Z}}_g = \{h_{\phi^*}(x) | x \in \bar{\mathcal{X}}_g\}$ is the set of output features for $\bar{\mathcal{X}}_g$.

Proof. Let $\mu_t^c(x)$ and $\sigma_t^c(x)$ denote the posterior mean and standard deviation of $\mathcal{GP}^{\text{cold}}$ at the end of round t after observing $\mathcal{D}_{t-1}^{\text{cold}} = \{(x_i, y_i)_{i=1}^{t-1}\}$. Similarly, we will use $\mu_t^w(x)$ and $\sigma_t^w(x)$ to denote the posterior mean and standard deviation of $\mathcal{GP}^{\text{warm}}$ at the end of round t after observing $\mathcal{D}_{t-1}^{\text{warm}} = \{(h_\phi(x_i), y_i)_{i=1}^{t-1}\}$.

Lemma 2. *Pick $\delta \in (0,1)$ and set $\beta_t = 2\log\left(\frac{|\mathcal{X}|\pi_t}{\delta}\right)$ where $\Sigma_{t\geq 1}\pi_t^{-1} = 0.5$, $\pi_t > 0$ (e.g. $\pi_t = \frac{\pi^2 t^2}{3}$). Define $\mu_t(x) = \lambda_t(x)\mu_t^c(x) + (1-\lambda_t(x))\mu_t^w(x)$ and $\sigma_t(x) = \lambda_t(x)\sigma_t^c(x) + (1-\lambda_t(x))\sigma_t^w(x)$. Then,*

$$P\{|f(x) - \mu_t(x)| \leq \beta_t^{1/2}\sigma_t(x), \forall x \in \mathcal{X}, \forall t \geq 1\} \geq 1-\delta.$$

Proof. Fix $t \geq 1$ and $x \in \mathcal{X}$. Based on Assumption 1, conditioned on $\mathcal{D}_{t-1}^{\text{cold}}$, $f(x) \sim \mathcal{N}(\mu_t^c(x), \sigma_t^c(x))$. Similarly, Assumption 2 implies that conditioned on $\mathcal{D}_{t-1}^{\text{warm}}$, $f(x) \sim \mathcal{N}(\mu_t^w(x), \sigma_t^w(x))$. Let $\mathcal{A}$ be the event that $|f(x) - \mu_t^c(x)| \leq \beta_t^{1/2}\sigma_t^c(x)$ and $\mathcal{B}$ the event that $|f(x) - \mu_t^w(x)| \leq \beta_t^{1/2}\sigma_t^w(x)$. From proof of Lemma 5.1 in [118] we know that given a normal distribution $z \sim \mathcal{N}(0,1)$, $P\{z > c\} \leq 0.5e^{-\frac{c^2}{2}}$. Using $z = \frac{f(x)-\mu_t^c(x)}{\sigma_t^c(x)}$ and $c = \beta_t^{1/2}$, $P\{\bar{\mathcal{A}}\} \leq e^{-\beta_t/2}$. Similarly, $P\{\bar{\mathcal{B}}\} \leq e^{-\beta_t/2}$. Using union bound, we have:

$$P\{\bar{\mathcal{A}} \vee \bar{\mathcal{B}}\} \leq P\{\bar{\mathcal{A}}\} + P\{\bar{\mathcal{B}}\} \leq 2e^{-\beta_t/2}.$$

By union bound, we have:

$$P\{\bar{\mathcal{A}} \vee \bar{\mathcal{B}}\} \leq |\mathcal{X}| \sum_{t\geq 1} 2e^{-\beta_t/2} \leq \delta \qquad \forall x \in \mathcal{X}, \forall t \geq 1.$$

The event in this Lemma is just $\mathcal{A} \wedge \mathcal{B}$ and the proof is concluded. □

Next, we state two lemmas from prior work.

Lemma 3. *If $|f(x) - \mu_t(x)| \leq \beta_t^{1/2}\sigma_t(x)$, then r_t is bounded by $2\beta_t^{1/2}\sigma_t(x_t)$.*

Proof. See Lemma 5.2 in [118]. It employs the results of Lemma 2 to prove the statement. □

Lemma 4. *Let $\sigma_t^2(x)$ denote the posterior variance of a GP after $t-1$ observations, and let $A \subset \mathcal{X}$. Assume that we have queried f at n points $(x_t)_{t=1}^n$ of which s points are in A. Then $\sum_{t:x_t \in A} \sigma_t^2(x) \leq \frac{2}{\log(1+\sigma^{-2})} \Psi_s(A)$.*

Proof. See Lemma 8 in [58]. □

Proof for Theorem 1 From Lemma 3, we have:

$$r_t^2 \leq 4\beta_t \sigma_t^2(x_t) \tag{6.5}$$

Summing over instantaneous regrets for T rounds, we get:

$$\begin{aligned}
\sum_{t=1}^{T} r_t^2 &\leq \sum_{t=1}^{T} 4\beta_t \sigma_t^2(x_t) && (6.6)\\
&\leq 4\beta_T \sum_{t=1}^{T} \sigma_t^2(x_t) && (6.7)\\
&\leq 4\beta_T \left(\sum_{t:x_t \in \chi_g} \sigma_t^{c^2}(x_t) + \sum_{t:x_t \in \bar{\chi}_g} \sigma_t^{w^2}(x_t) \right) && (6.8)\\
&\leq \frac{8\beta_T}{1+\sigma_n^{-2}} \left(\Psi_{T-s}(\chi_g) + \Psi_s(\bar{\mathcal{Z}}_g) \right) && (6.9)
\end{aligned}$$

Eq 6.7 follows from the monotonicity of $\beta_t = 2\log(\pi_t/\delta)$. Eq. 6.8 follows from the definition of σ_t in Lemma 2 and the last inequality in Eq. 6.9 follows from Lemma 4.

Finally, from Cauchy-Schwartz inequality, we know that $R_T^2 \leq T \sum_{t=1}^{T} r_t^2$. Combining with Eq. 6.9, we obtain the result in Theorem 1.

□

Extension to Continuous Domains

We will now derive regret bounds for the general case where $\mathcal{X} \subset [0, r]^d$ is a d-dimensional compact and convex set with $r > 0$. This will critically require an additional Lipschitz continuity assumption on f.

Theorem 5. *Suppose that kernels κ^c and κ^w are such that the derivatives of $\mathcal{GP}^{cold}$ and $\mathcal{GP}^{warm}$ sample paths are bounded with high probably. Precisely, for some constants $a, b > 0$,*

$$P\left\{ \left| \sup_{x \in \mathcal{X}} \frac{\partial f}{\partial x_j} \right| > L \right\} \leq a e^{-(L/b)^2}, j = 1, 2, \ldots, d. \tag{6.10}$$

Pick $\delta \in (0,1)$, and set $\beta_t = 2\log(4\pi^2 t^2/3\delta) + 4d\log(dtbr\sqrt{\log(4da/\delta)})$. Then, running JUMBO for T iterations results in a sequence of candidates $(x_t)_{t=1}^{t=T}$ for which the following holds with probability at least $1-\delta$:

$$R_T < \sqrt{CT\beta_T\{\Psi_{T-s}(\chi_g) + \Psi_s(\bar{\mathcal{Z}}_g)\}} + \frac{\pi^2}{6}, \forall T \geq 1$$

where $C = {}^1/_{(1+\sigma_n^2)}$.

To start the proof, we first show that we have confidence on all the points visited by the algorithm.

Lemma 6. *Pick $\delta \in (0,1)$ and set $\beta_t = 2\log(\pi_t/\delta)$, where $\sum_{t\geq 1}\pi_t^{-1} = 0.5$, $\pi_t > 0$. Define $\mu_t(x) = \lambda_t(x)\mu_t^c(x) + (1-\lambda_t(x))\mu_t^w(x)$ and $\sigma_t(x) = \lambda_t(x)\sigma_t^c(x) + (1-\lambda_t(x))\sigma_t^w(x)$. Then,*

$$|f(x_t) - \mu_t(x_t)| \leq \beta_t^{1/2}\sigma_t(x_t),\ \forall t \geq 1$$

holds with a probability of at least $1-\delta$.

Proof. Fix $t \geq 1$ and $x \in \chi$. Similar to Lemma 3, $P\{\bar{\mathcal{A}} \vee \bar{\mathcal{B}}\} \leq 2e^{-\beta_t/2}$. Since $e^{-\beta_t/2} = \delta/\pi_t$, using the union bound for $t \geq 1$ concludes the statement. □

For the purpose of analysis, we define a discretization set $\chi_t \subset \chi$, so that the results derived earlier can be re-applied to bound the regret in continuous case. To enable this approach we will use conditions on L-Lipschitz continuity to obtain a valid confidence interval on the optimal solution x^*. Similar to [118], let us choose discretization χ_t of size τ_t^d (i.e. τ_t uniformly spaced points per dimension in χ) such that for all $x \in \chi$ the closest point to x in χ_t, $[x]_t$, has a distance less than some threshold. Formally, $||x - [x]_t||_1 \leq {}^{rd}/_{\tau_t}$.

Lemma 7. *Pick $\delta \in (0,1)$ and set $\beta_t = 2\log(4\pi_t/\delta) + 4d\log(dtbr\sqrt{\log(4da/\delta)})$, where $\sum_{t\geq 1}\pi_t^{-1} = 0.5$, $\pi_t > 0$. Then, for all $t \geq 1$, the regret is bounded as follows:*

$$r_t \leq 2\beta_t^{1/2}\sigma_t(x_t) + \frac{1}{t^2} \tag{6.11}$$

with probabilty of at least $1-\delta$.

Proof. In light of Lemma 6,the proof follows directly from Lemma 5.8 in [118]. □

Proof of Theorem 5

Proof. From Eq. 6.9 in the proof of Theorem 1, we have shown that:

$$\sum_{t=1}^{T} 4\beta_t\sigma_t^2(x_t) \leq C\beta_T(\Psi_{T-s}(\chi_g) + \Psi_s(\bar{\mathcal{Z}}_g)) \qquad \forall T \geq 1.$$

Therefore, using Cauchy-Schwarz:

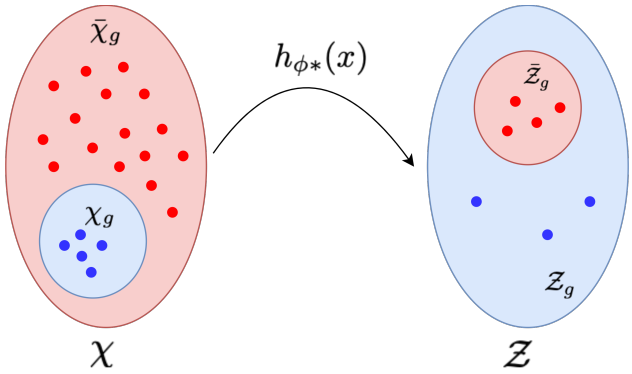


Figure 6.2: The effect of the pre-trained NN $h_{\phi^*}(x)$ on $\mathcal{X}$. In the desirable case, $\bar{\mathcal{X}}_g$ gets significantly compressed to $\bar{\mathcal{Z}}_g$.

$$\sum_{t=1}^{T} 2\beta_t^{1/2}\sigma_t(x_t) \leq \sqrt{C\beta_T(\Psi_{T-s}(\mathcal{X}_g) + \Psi_s(\bar{\mathcal{Z}}_g))} \qquad \forall T \geq 1.$$

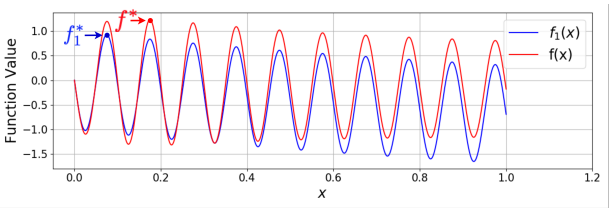
Since $\sum_{t=1}^{T} \frac{1}{t^2} \leq \frac{\pi^2}{6}$, Theorem 5 follows Lemma 7. □

Based on the regret bound in Eq. 6.4, we can conclude that if the partitioning $\mathcal{X}_g$ is chosen such that $\Pi(\bar{\mathcal{Z}}_g) \ll \Pi(\bar{\mathcal{X}}_g)$ *and* $\Pi(\mathcal{X}_g) \ll \Pi(\mathcal{X})$, then JUMBO has a tighter bound than GP-UCB. The first condition implies that the second term in Eq. 6.4 is negligible and intuitively means that $\mathcal{GP}^{\text{warm}}$ will only need a few samples to infer the posterior of f defined on $\bar{\mathcal{X}}_g$, making BO more sample efficient. The second condition implies that the $\Psi_{T-s}(\mathcal{X}_g) \ll \Psi_T(\mathcal{X})$ which in turn makes the regret bound of JUMBO tighter than GP-UCB. Note that $\mathcal{X}_g$ cannot be made arbitrarily small, since $\bar{\mathcal{X}}_g$ (and therefore $\bar{\mathcal{Z}}_g$) will get larger which conflicts with the first condition.

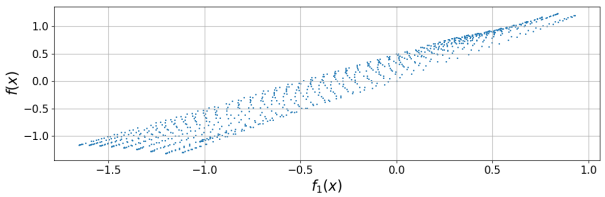
Figure 6.2 provides an illustrative example. If the learned feature space $h_{\phi^*}(x)$ compresses set $\bar{\mathcal{X}}_g$ to a smaller set $\bar{\mathcal{Z}}_g$, then $\mathcal{GP}^{\text{warm}}$ can infer the posterior of $g(h_{\phi^*}(x))$ with only a few samples in $\bar{\mathcal{X}}_g$ (because MIG is lower). Such $h_{\phi^*}(x)$ will likely emerge when tasks share high-level features with one another. In the appendix, we have included an empirical analysis to show that $\mathcal{GP}^{\text{warm}}$is indeed operating on a compressed space $\mathcal{Z}$. Consequently, if $\mathcal{X}_g$ is reflective of promising regions consisting of near-optimal points i.e. $\mathcal{X}_g = \{x \in \mathcal{X} \mid f(x^*) - f(x) \leq l_f\}$ for some $l_f > 0$, BO will be able to quickly discard points from subset $\bar{\mathcal{X}}_g$ and acquire most of its points from $\mathcal{X}_g$.

Choice of interpolation coefficient $\lambda_t(x)$

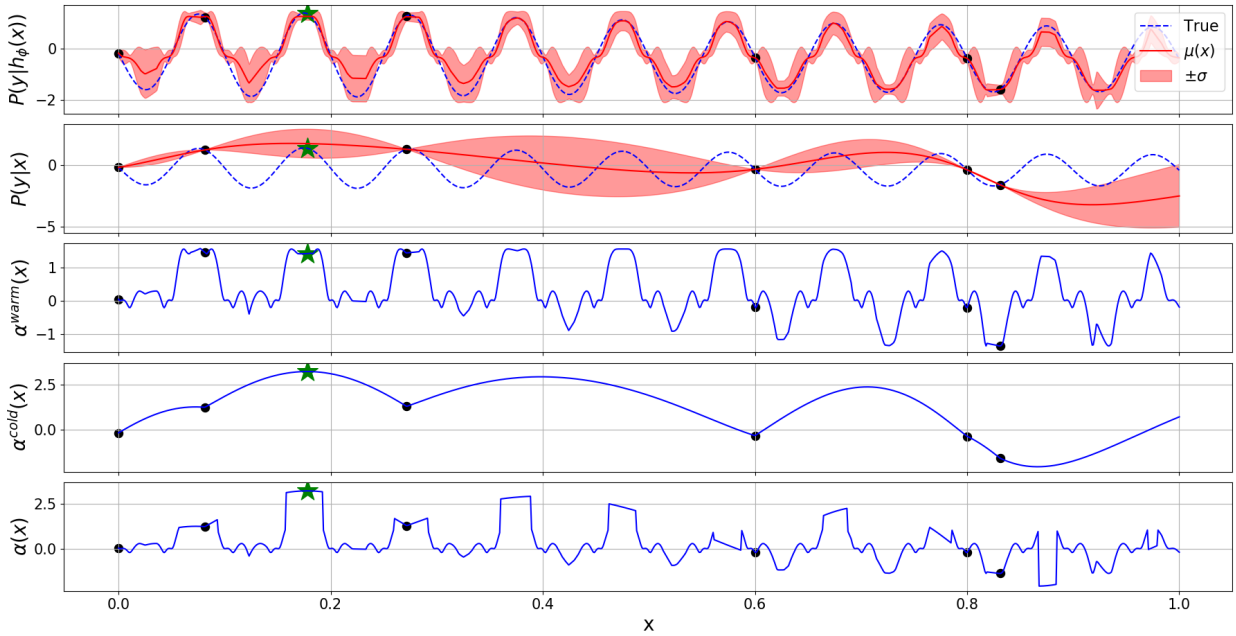
The above discussion suggests that the partitioning $\mathcal{X}_g$ should ideally consist of near-optimal points. In practice, we do not know f and hence, we rely on our surrogate model to define $\mathcal{X}_g^{(t)} = \{x \in \mathcal{X} \mid \alpha_t^{\text{warm}*} - \alpha_t^{\text{warm}}(x) \leq l_\alpha\}$. Here, $\alpha_t^{\text{warm}*}$ is the optimal value of $\alpha_t^{\text{warm}}(x)$ and the



(a) Target (red) and auxiliary (blue) task



(b) Correlation between target and auxiliary tasks



(c) Posterior of $\mathcal{GP}^{\text{warm}}$, $\mathcal{GP}^{\text{cold}}$, their UCB and JUMBO's acquisition function

Figure 6.3: Dynamics of JUMBO after observing 6 data points (a) The two functions have different optimums (b) The tasks are related (c) Iteration 4 of the BO with our proposed model, from top to bottom: (1) GP modeling input to objective using $(x, h_{\phi^*}(x), y)$ samples (2) GP modeling input to objective using (x, y) samples (3) UCB acquisition function for $\mathcal{GP}^{\text{warm}}$(4) UCB acquisition function for $\mathcal{GP}^{\text{cold}}$(5) JUMBO's acquisition function that compromises between the optimum of the two.

acquisition threshold $l_\alpha > 0$ is a hyper-parameter used for defining near-optimal points w.r.t. $\alpha_t^{\text{warm}}(x)$. At one extreme, $l_\alpha \to \infty$ corresponds to the case where $\alpha_t(x) = \alpha_t^{\text{cold}}(x)$ (i.e. the GP-UCB routine) and the other extreme $l_\alpha \to 0$ corresponds to case with $\alpha_t(x) = \alpha_t^{\text{warm}}(x)$.

Figure 6.3 illustrates a synthetic 1D example of how JUMBO obtains the next query point. Figure 6.3a shows the main objective $f(x)$ (red) and the auxiliary task $f_1(x)$ (blue). They share a periodic structure but have different optimums. Figure 6.3b shows the correlation between the two.

Applying GP-UCB [118] will require a considerable amount of samples to learn the periodic structure and the optimal solution. However in JUMBO, as shown in Figure 6.3c, the warm-GP, trained on $(h_{\phi^*}(x), y)$ samples, can learn the periodic structure using only 6 samples, while the posterior of the cold-GP has not yet learned this structure.

It can also be noted from Figure 6.3c that JUMBO's acquisition function is $\alpha_t^{\text{cold}}(x)$ when the value of $\alpha_t^{\text{warm}}(x)$ is close to $\alpha_t^{\text{warm}*}$. Therefore, the next query point (marked with a star) has a high score based on both acquisition functions. We summarize JUMBO in Algorithm 4.

Algorithm 3 JUMBO

Input: Offline auxiliary dataset $\mathcal{D}^{\text{aux}}$, Offline target dataset $\mathcal{D}_0^f$ (optional; default: empty set), Threshold l_α
Output: Sequence of solution candidates $\{x_t\}_{t=1}^T$ maximizing target function f
Initialize NN $h_\phi(x)$, $\mathcal{GP}^{\text{cold}}$, $\mathcal{GP}^{\text{warm}}$.
Pretrain NN params jointly with $\mathcal{GP}^{\text{cold}}$ and $\mathcal{GP}^{\text{warm}}$ hyper-params using $\mathcal{D}^{\text{aux}}$ and $\mathcal{D}_0^f$ as per Eq. 7.3.
Initialize $\mathcal{D}_0^{\text{cold}} = \{\}$, $\mathcal{D}_0^{\text{warm}} = \{\}$.
for round $t = 1 \leftarrow T$ **do**
    Set $\alpha_t^{\text{warm}*} = \arg\max_{x \in \chi} \alpha_t^{\text{warm}}(x)$.
    Set $\lambda_t(x) = \mathbb{1}(\alpha_t^{\text{warm}*} - \alpha_t^{\text{warm}}(x) \leq l_\alpha)$.
    Set $\alpha_t(x) = \lambda_t(x)\alpha_t^{\text{cold}}(x) + (1 - \lambda_t(x))\alpha_t^{\text{warm}}(x)$
    Pick $x_t = \arg\max_{x \in \chi} \alpha_t(x)$.
    Obtain noisy observation y_t for x_t.
    Update $\mathcal{D}_t^{\text{cold}} \leftarrow \mathcal{D}_{t-1}^{\text{cold}} \cup \{(x_t, y_t)\}$ and $\mathcal{GP}^{\text{cold}}$.
    Update $\mathcal{D}_t^{\text{warm}} \leftarrow \mathcal{D}_{t-1}^{\text{warm}} \cup \{(h_{\phi^*}(x_i), y_i)\}$ and $\mathcal{GP}^{\text{warm}}$.

6.5 Experiments

We are interested in investigating the following questions: (1) How does JUMBO perform on benchmark real-world black-box optimization problems relative to baselines? (2) How does the choice of threshold l_α impact the performance of JUMBO? (3) Is it necessary to have a non-linear mapping on the features learned from the offline dataset or a BLR layer is sufficient?

Automated Hyperparameter Tuning

Datasets. We consider the task of optimizing hyperparameters for fully-connected NN architectures on 4 regression benchmarks from HPOBench [60]: *Protein Structure* [102], *Parkinsons Telemonitoring* [123], *Naval Propulsion* [15], and *Slice Localization* [35].

HPOBench provides a look-up-table-based API for querying the validation error of all possible hyper-parameter configurations for a given regression task. These configurations

are specified via 9 hyperparameters, that include continuous, categorical, and integer-valued variables.

The objective we wish to minimize is the validation error of a regression task after 100 epochs of training. For this purpose, we consider an offline dataset that consists of validation errors for some randomly chosen configurations after 3 epochs on a given dataset. The target task is to optimize this error after 100 epochs. In [60], the authors show that this problem is non-trivial as there is a small correlation between epochs 3 and 100 for top-1% configurations across all datasets of interest.

We handle categorical and integer-valued variables in BO similar to [31]. In particular, we used $\kappa^c(T(x), T(x'))$ as the kernel where $T : \mathcal{X} \to \mathcal{T}$ is a deterministic transformation that maps the continuous optimization variable x to a representation space $\mathcal{T}$ that adheres to a meaningful distance measurement. For example, for categorical parameters, it converts a continuous input to a one-hot encoding corresponding to a choice for that parameter, and for integer-valued variables, it converts the continuous variable to the closest integer value. Similarly, for the pre-training phase, we also train using $h_\phi(T(x))$.

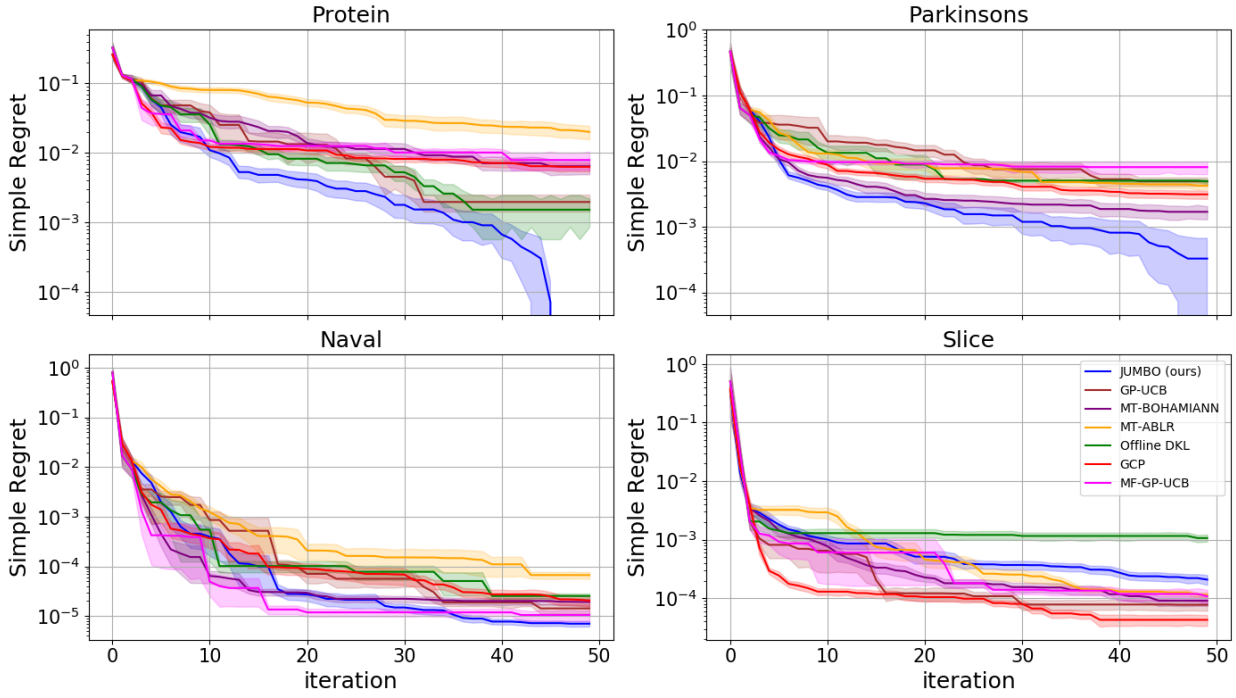


Figure 6.4: The regret of MBO algorithms on Protein, Parkinsons, Naval, and Slice datasets. Standard errors are measured across 20 random seeds.

Evaluation protocol. We validate the performance of JUMBO against the following baselines with a UCB acquisition function [118]:

- **GP-UCB** [118] (i.e. cold-GP only) trains a GP from scratch disregarding $\mathcal{D}^{\text{aux}}$ completely. Equivalently, it can be interpreted as JUMBO with $\lambda_t(x) = 1 \ \ \forall x, t \geq 1$ in Eq. 6.2 and $\alpha(x) = \alpha^{\text{UCB}}(x)$.
- **MT-BOHAMIANN** [116] trains a BNN on all tasks jointly via SGHMC (Section 6.1).
- **MT-ABLR** [89] trains a shared NN followed by task-specific BLR layers (Section 6.1).
- **GCP** [107] uses Gaussian Copula Processes to jointly model the offline and online data.
- **MF-GP-UCB** [58] extends the GP-UCB baseline to a multi-fidelity setting where the source task can be interpreted as a low-fidelity proxy for the target task.
- **Offline DKL** (i.e. warm-GP only) is our proposed extension to Deep Kernel Learning, where we train a single GP online in the latent space of a NN pre-trained on $\mathcal{D}^{\text{aux}}$ (See Section 6.2 for details). Equivalently, it can be interpreted as JUMBO with $\lambda_t(x) = 0$ in Eq. 6.2.

Results. We run JUMBO (with $l_\alpha = 0.1$) on all baselines for 50 rounds and 5 random seeds each and measure the simple regret per iteration.

The regret curves are shown in Figure 6.4. We find that JUMBO achieves lower regret than the previous state-of-the-art algorithms for MBO in almost all cases.

We believe the slightly worse performance on the slice dataset relative to other baselines is due to the extremely low top-1% correlation between epoch 3 and epoch 100 on this dataset as compared to others (See Figure 10 in [60]), which could result in a suboptimal search space partitioning obtained via the warm-GP. For all other datasets, we find JUMBO to be the best performing method. Notably, on the Protein dataset, JUMBO is always able to find the global optimum, unlike the other approaches.

Automated Circuit Design

Next, we consider the two-stage operational amplifier design example that we have introduced in chapter 4.

In practice, we are interested in performing *layout* simulations for measuring the performance metric. To emulate the mismatch between layout and schematic simulations we created a different `NGSPICE` template that includes estimated parasitics that scale as a function of the transistor sizes. So that up-sizing transistors would not come for free in the simulations.

We then use the simulation engine with no parasitics for offline data collection of all the specification metrics outlined in table 4.1. From those metrics we choose three as the auxiliary signals to model using the pre-trained neural network: cost, gain, settling time. After pre-training the neural network, the frozen embedding is used for optimizing the score function of the circuit using the parasitic-aware simulation engine. This experimental setup ensures that the auxiliary dataset is not entirely aligned with the target objective's performance. For the offline dataset, we collected 1000 pairs of circuit configurations and the three auxiliary signals.

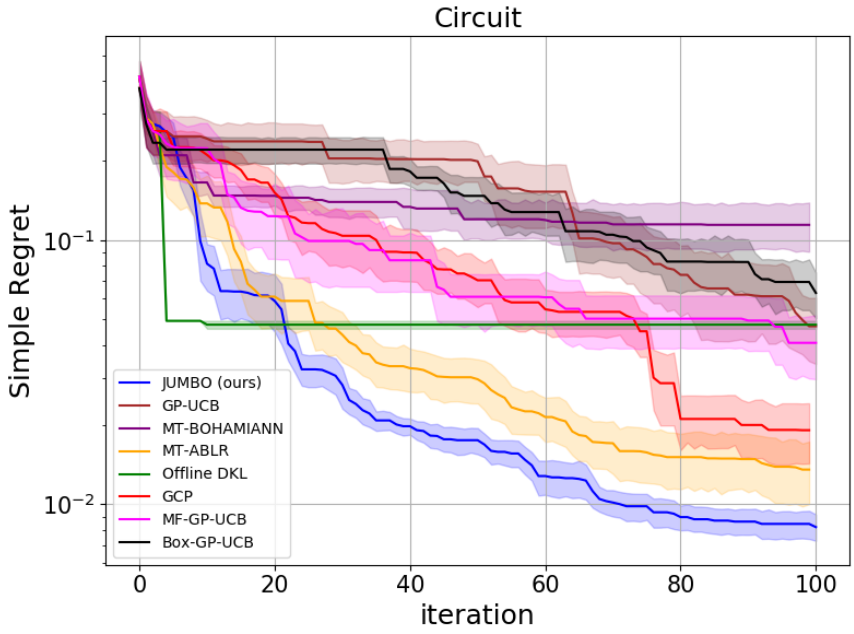


Figure 6.5: Circuit Design results

We consider the same baselines as before. We also consider BOX-GP-UCB [88] which confines the search space to a hyper-cube over the promising region based on all auxiliary tasks in the offline data. Unlike the considered HPO problems, the offline circuit dataset contains data from more than just one auxiliary task, allowing us to consider BOX-GP-UCB as a viable baseline. MF-GP-UCB was run only with the schematic score as the lower fidelity approximation of the target function. We ran each algorithm with $l_\alpha = 0.1$ for 100 iterations and measured simple regret against iteration. As reflected in the regret curves in Figure 6.5, JUMBO outperforms other algorithms.

6.6 Ablations

Choice of l_α. The threshold l_α is a key design hyperparameter for defining the acquisition function in JUMBO. JUMBO with $l_\alpha = \infty$ reduces to GP-UCB (i.e. cold-GP only) and with $l_\alpha = 0$, it reduces to offline DKL (i.e. warm-GP only).

Table 6.1 shows the effect of different choices for l_α on the performance of the algorithm for both HPO and circuit design problems. As we can see, small values for l_α (e.g. ~ 0.01) cause JUMBO to rely more on the accuracy of the warm-GP model and result in sub-optimal convergence in case of model discrepancy between warm-GP and the target task. On the other hand, bigger values of l_α (e.g. ~ 0.2) cause JUMBO to give more weight to the cold-GP and rely less on prior data. We also note that there is a wide range of l_α that JUMBO

performs well relative to other baselines, suggesting a good degree of robustness and less tuning in practice. Even though the optimal choice of l_α depends on the exact problem setup (e.g. 0.05 for circuits problem, and 0.2 for Slice localization), we have found that the choice of $l_\alpha = 0.1$ is a good initial choice for all the problems considered.

Table 6.1: The average normalized simple regret at the last iteration for different values of l_α (lower is better). The scores are normalized to GP-UCB's simple regret at the last iteration.

	GP-UCB	JUMBO-0.01	JUMBO-0.05	JUMBO-0.1	JUMBO-0.2	Offline DKL ($l_\alpha = \infty$)
Protein	1.0 ± 0.08	2.09 ± 0.00	1.45 ± 0.18	**0.00 ± 0.00**	0.29 ± 0.05	0.77 ± 0.13
Parkinsons	1.0 ± 0.05	0.52 ± 0.07	0.19 ± 0.05	**0.07 ± 0.05**	0.45 ± 0.05	1.0 ± 0.02
Naval	1.0 ± 0.07	0.97 ± 0.07	**0.46 ± 0.03**	0.49 ± 0.04	0.78 ± 0.07	1.78 ± 0.06
Slice	1.0 ± 0.02	5.54 ± 0.5	2.87 ± 0.12	2.69 ± 0.29	**0.94 ± 0.17**	13.77 ± 0.57
Circuit	1.0 ± 0.06	0.16 ± 0.00	**0.09 ± 0.00**	0.18 ± 0.01	0.24 ± 0.01	1.01 ± 0.01

Effect of auxiliary tasks. It is important to analyze how learning on other tasks affects performance. To this end, we considered the circuit design problem with 1 and 3 auxiliary offline tasks. In Figure 6.6, task 1 (yellow) is the most correlated and task 3 (red) is the least correlated task with the objective function. The regret curves suggest that the performance would be poor if the correlation between tasks is low. Moreover, the features pre-trained on the combination of all three tasks provide more information to the warm-GP than those pre-trained only on one of the tasks.

BLR with JUMBO's acquisition function. A key difference between JUMBO and ABLR [89] is replacing the BLR layer with a GP. To show the merits of having a GP, we ran an experiment on the Protein dataset and replaced the GP with a BLR in JUMBO's procedure. Figure 6.7 shows that JUMBO with $\mathcal{GP}^{\text{warm}}$significantly outperforms JUMBO with a BLR layer.

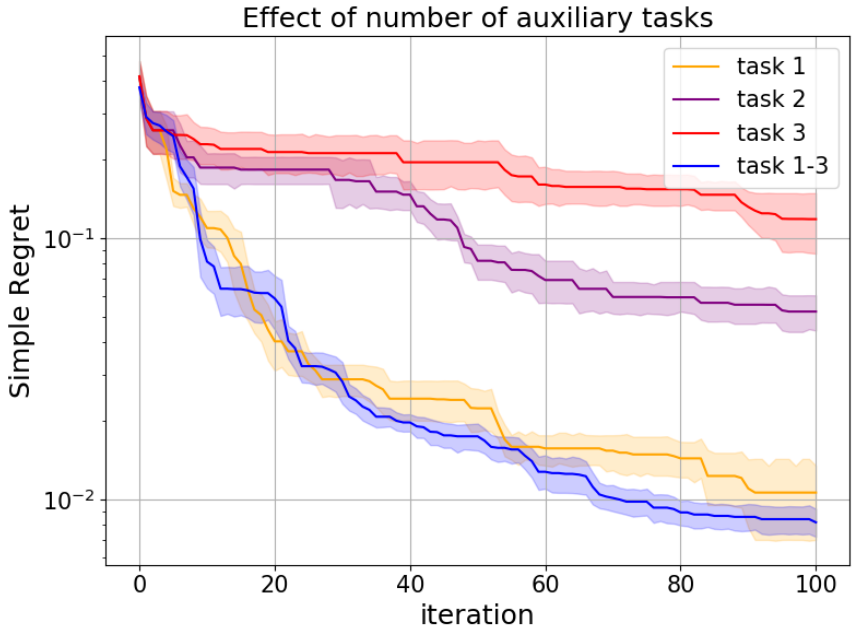


Figure 6.6: JUMBO with 3 aux. tasks is better than JUMBO with each individual aux. tasks.

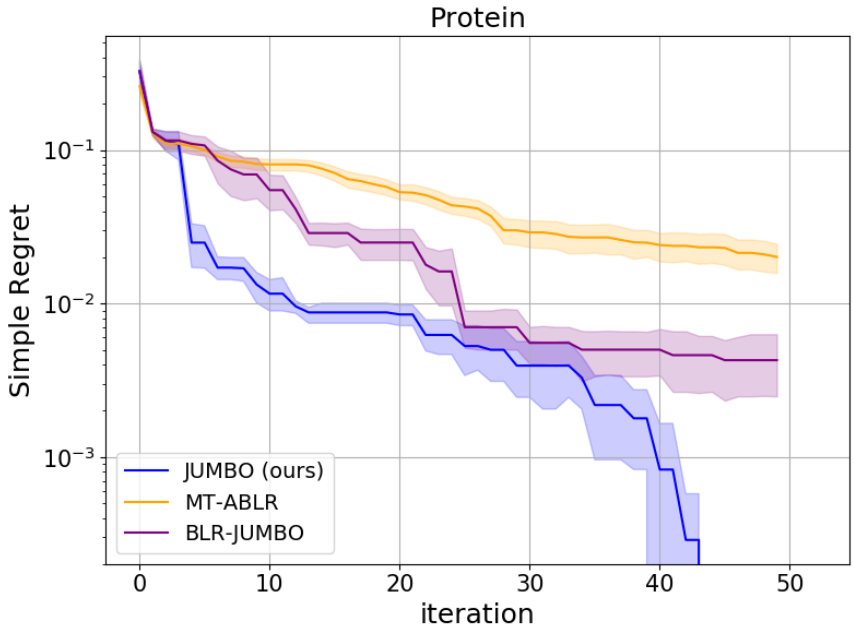


Figure 6.7: The Non-linear mapping is a crucial piece of JUMBO's algorithm.

Space compression through the pre-trained NN. In this experiment we studied the latent space of a NN fed with uniformly sampled inputs for circuit design and see that 75% of data variance is preserved in only 4 dimensions (with $n_z = 32$), suggesting that the warm-GP is operating in a compressed space.

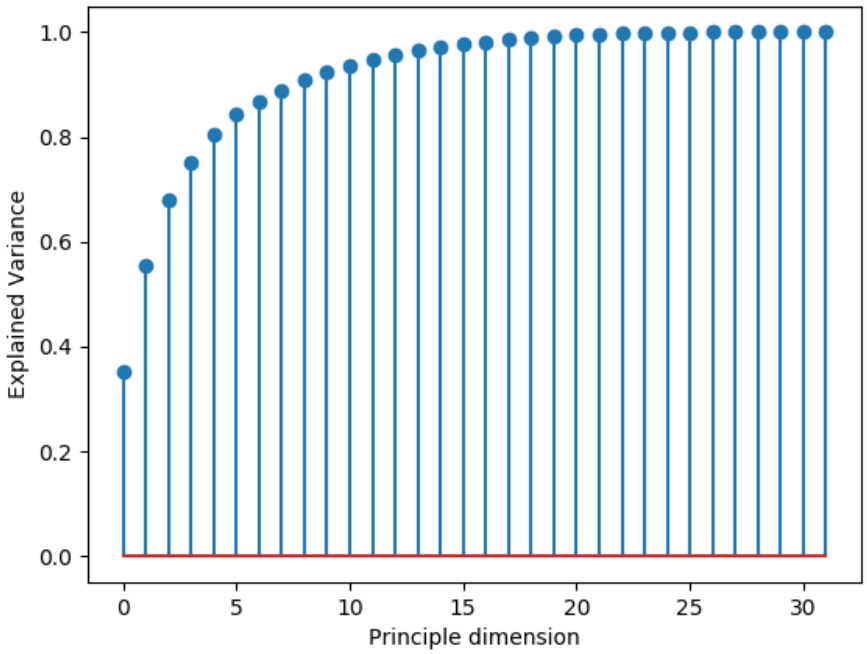


Figure 6.8: Explained Variance of latent dimensions

Dynamic choice of λ_t. In this ablation, we illustrate that the dynamic choice of λ_t is indeed better than choosing it to be a constant value. The intuition behind it is that by choosing a constant coefficient we essentially allow the acquisition function to choose points with very high α^{cold} but low α^{warm} scores. However, α^{cold} should not be trusted because of the warm-start problem in BO.

Figure 6.9 compares JUMBO with dynamic and constant λ_t on the four HPO problems. It can be seen that JUMBO with constant $\lambda_t = 0.5$ immaturely reaches a sub-optimal solution in all the experiments.

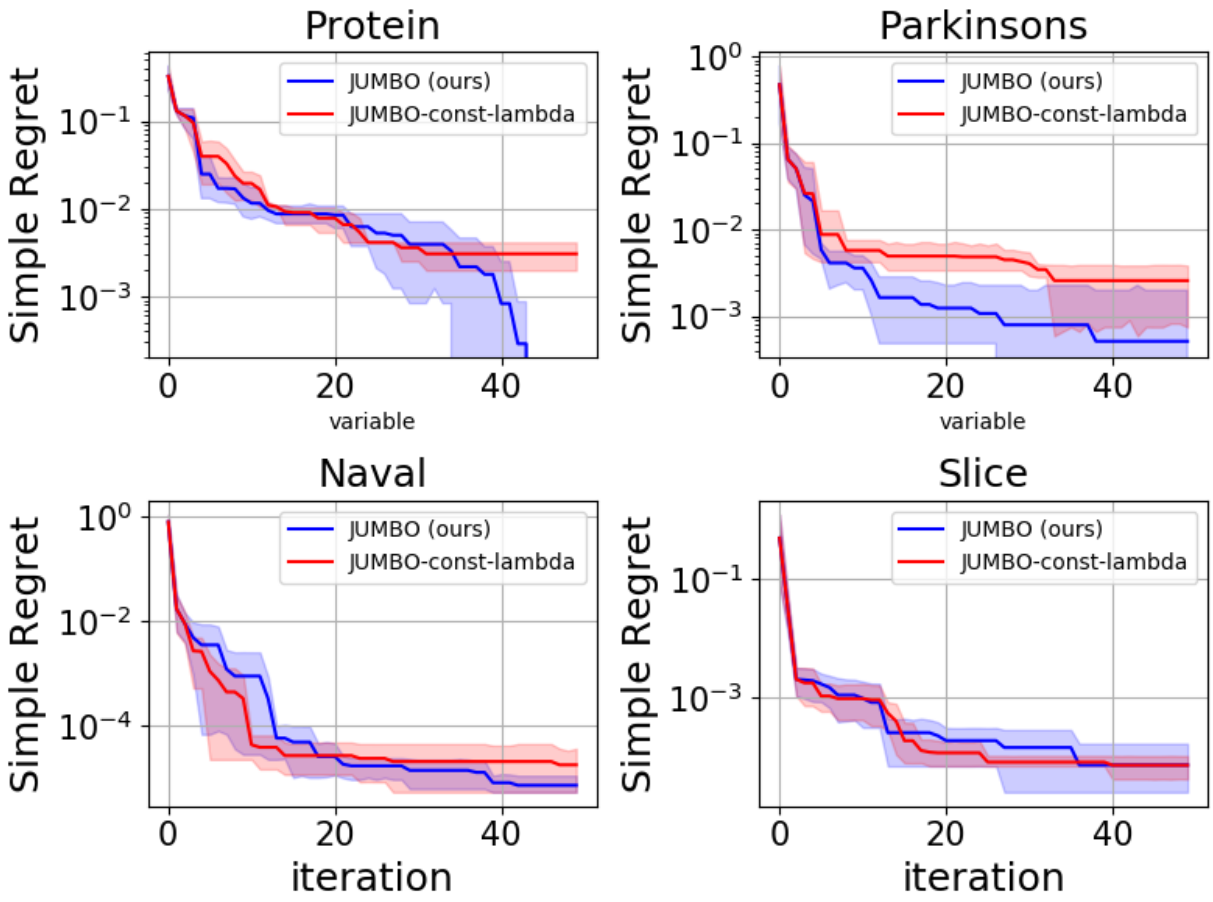


Figure 6.9: Ablation on the dynamic choice of λ_t

6.7 Implementation Details

Neural Network Architecture

Figure 6.10 illustrates the skeleton of the architecture that was used for all experiments. The input configuration is fed to a multi-layer perceptron of n_l layers with n_u hidden units. Then, optionally, a dropout layer is applied to the output and the result is fed to another non-linear layer with n_z outputs. The latent features are then mapped to the output with a linear layer. All activations are tanh.

For HPO experiments, we have used $n_u = 32, n_z = 4, n_l = 3$, learning rate $= 5 \times 10^{-5}$, and batch size $= 128$. For circuit experiments we used $n_u = 200, n_z = 32, n_l = 3$, learning rate $= 3 \times 10^{-4}$, batch size $= 64$, and dropout rate of 0.5. These hyper-parameters were chosen based on random search by observing the prediction accuracy of the pre-training model on

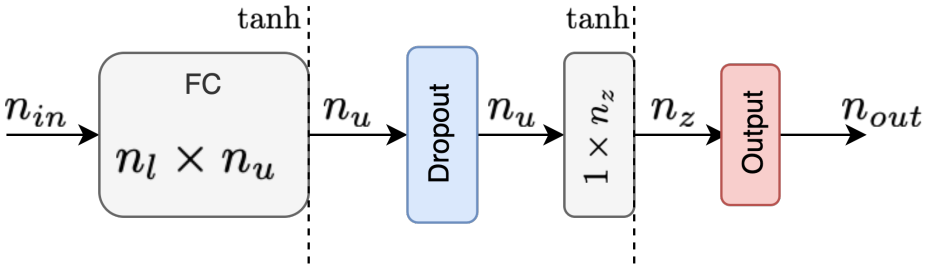


Figure 6.10: NN Architecture during pre-training: The first blocks is n_l layers of n_u hidden units with tanh activations. Following that is a dropout layer and then a single layer perceptron to get n_z features. Thereafter, the latent features are mapped linearly to the output.

the auxiliary validation dataset which was 20% of the overall dataset.

Details of training the Gaussian Process hyper-parameters

For both warm and cold GP, we consider a Matern kernel (i.e. $\kappa(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}r^2)K_\nu(\sqrt{2\nu}r^2)$ with $\nu = 2.5$ where $r^2 = \frac{||x-x'||^2}{\theta}$). The length scale θ and observation noise σ_n are optimized in every iteration of BO by taking 100 gradient steps on the likelihood of observations via Adam optimizer with a learning rate of 0.1.

Acquisition Function Details

For all experiments, we used Upper Confidence Bound with the exploration-exploitation hyper-parameter at round t set as $10 \exp \frac{-t}{T}$ where T is the budget of the total number of iterations. This way we favor exploration more initially and gradually drift to more exploitation as we approach the end of the budget. For optimization of the acquisition function, we use the derivative-free algorithm CMA-ES [42].

6.8 Conclusion

We proposed JUMBO, a no-regret algorithm that employs a careful hybrid of neural networks, Gaussian Processes, and a novel acquisition procedure for scalable and sample-efficient Multi-task Bayesian Optimization. We derived JUMBO's theoretical regret bound and empirically showed it outperforms other competing approaches on a set of real-world optimization problems. We specifically showed how this combination of pre-training and Bayesian optimization can result in improved sample efficiency in the application of analog circuit design. Our method scales gracefully ($O(n)$) with the number of data points in the auxiliary datasets and therefore can be scaled up in the pre-training phase. For future directions it would be interesting to see whether the graph neural network models pre-trained in Chapter

5 can be re-used as the feature extractor for JUMBO to improve the sample efficiency of Bayesian optimization methods.

Chapter 7

Few-shot Imitation Learning via Skill Extraction Pre-training

In the previous chapters, we covered how we can use pre-training to improve the sample efficiency of black-box optimization methods. However, there are other fields of AI that could also benefit from the paradigm of pre-training. For instance, the domains of robotics, (motor) control, and reinforcement learning (RL) have largely operated under a task-specific and tabula rasa learning paradigm, thereby limiting the generalization and scale. In this chapter, we take a step in the direction of using offline existing trajectory datasets to pre-train models or policies that can be used for unseen downstream control tasks.

A desirable property of autonomous agents is the ability to both solve long-horizon problems and generalize to unseen tasks. Recent advances in data-driven skill learning have shown that extracting behavioral priors from offline data can enable agents to solve challenging long-horizon tasks with reinforcement learning. However, generalization to tasks unseen during behavioral prior training remains an outstanding challenge. To this end, we present Few-shot Imitation with Skill Transition Models (FIST), an algorithm that extracts skills from offline data and utilizes them to generalize to unseen tasks given a few downstream demonstrations. FIST learns an inverse skill dynamics model, a distance function, and utilizes a semi-parametric approach for imitation. We show that FIST is capable of generalizing to new tasks and substantially outperforms prior baselines in navigation experiments requiring traversing unseen parts of a large maze and 7-DoF robotic arm experiments requiring manipulating previously unseen objects in a kitchen.

7.1 Introduction

We are interested in developing control algorithms that enable robots to solve complex and practical tasks such as operating kitchens or assisting humans with everyday chores at home. There are two general characteristics of real-world tasks – long-horizon planning and generalizability. Practical tasks are often long-horizon in the sense that they require a robot

to complete a sequence of subtasks. For example, to cook a meal a robot might need to prepare ingredients, place them in a pot, and operate the stove before the full meal is ready. Additionally, in the real world, many tasks we wish our robot to solve may differ from tasks the robot has completed in the past but require a similar skill set. For example, if a robot learned to open the top cabinet drawer it should be able to quickly adapt that skill to open the bottom cabinet drawer. These considerations motivate our research question: *how can we learn skills that enable robots to generalize to new long-horizon downstream tasks?*

Recently, learning data-driven behavioral priors has become a promising approach to solving long-horizon tasks. Given a large unlabeled offline dataset of robotic demonstrations solving a diverse set of tasks this family of approaches [114, 90, 1] extract behavioral priors by fitting maximum likelihood expectation latent variable models to the offline dataset. The behavioral priors are then used to guide a Reinforcement Learning (RL) algorithm to solve downstream tasks. By selecting skills from the behavioral prior, the RL algorithm can explore in a structured manner and can solve long-horizon navigation and manipulation tasks. However, the generalization capabilities of RL with behavioral priors are limited since a different RL agent needs to be trained for each downstream task and training each RL agent often requires millions of environment interactions.

On the other hand, few-shot imitation learning has been a promising paradigm for generalization. In the few-shot imitation learning setting, an imitation learning policy is trained on an offline dataset of demonstrations and is then adapted, in few-shot, to a downstream task [21]. Few-shot imitation learning has the added advantage over RL in that it is often easier for a human to provide a handful of demonstrations than it is to engineer a new reward function for a downstream task. However, unlike RL with behavioral priors, few-shot imitation learning is most often limited to short-horizon problems. The reason is that imitation learning policies quickly drift away from the demonstrations due to error accumulation [105], and especially so in the few-shot setting when only a handful of demonstrations are provided.

While it is tempting to simply combine data-driven behavioral priors with few-shot imitation learning, it is not obvious how to do so since the two approaches are somewhat orthogonal. Behavioral priors are trained on highly multi-modal datasets such that a given state can correspond to multiple skills. Given a sufficiently large dataset of demonstrations for the downstream task the imitation learning algorithm will learn to select the correct mode. However, in the few-shot setting how do we ensure that during training on downstream data we choose the right skill? Additionally, due to the small sample size and long task horizon, a naive imitation learning policy will likely drift from the few-shot demonstrations. How do we prevent the imitation learning policy from drifting away from downstream demonstrations?

The focus of our work is the setup illustrated in Figure 1; we introduce Few-Shot Imitation Learning with Skill Transition Models (FIST), a new algorithm for few-shot imitation learning with skills that enables generalization to unseen but semantically similar long-horizon tasks to those seen during training. Our approach addresses the issues with skill selection and drifting in the few-shot setting with two main components. First, we introduce an

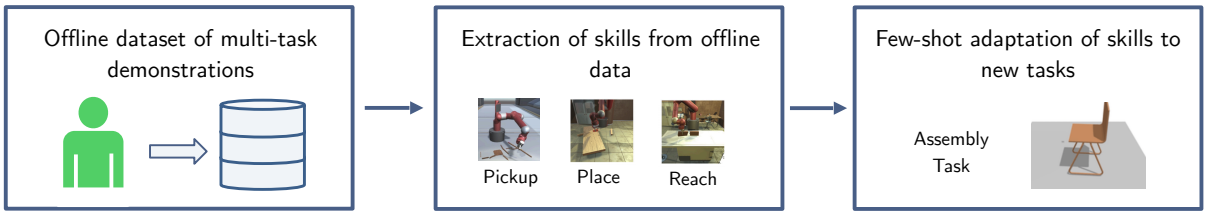


Figure 7.1: In this work we are interested in enabling autonomous robots to solve complex long-horizon tasks that were unseen during training. To do so, we assume access to a large multi-task dataset of demonstrations, extract skills from the offline dataset, and adapt those skills to new tasks that were unseen during training.

inverse skill dynamics model that conditions the behavioral prior not only on the current state but also on a future state, which helps FIST learn uni-modal future conditioned skill distribution that can then be utilized in few-shot. The inverse skill model is then used as a policy to select skills that will take the agent to the desired future state. Second, we train a distance function to find the state for conditioning the inverse skill model during evaluation. By finding states along with the downstream demonstrations that are closest to the current state, FIST prevents the imitation learning policy from drifting. We show that our method results in policies that can generalize to new long-horizon downstream tasks in navigation environments and multi-step robotic manipulation tasks in a kitchen environment. To summarize, we list our three main contributions:

1. We introduce FIST - an imitation learning algorithm that learns an inverse skill dynamics model and a distance function that is used for semi-parametric few-shot imitation.
2. We show that FIST can solve long-horizon tasks in both navigation and robotic manipulation settings that were unseen during training and outperforms previous behavioral prior and imitation learning baselines.
3. We provide insight into how different parts of the FIST algorithm contribute to final performance by ablating different components of our method such as future conditioning and fine-tuning on downstream data.

7.2 Related Work

Our approach combines ingredients from imitation learning and skill extraction to produce policies that can solve long-horizon tasks and generalize to tasks that are out of distribution but semantically similar to those encountered in the training set. We cover the most closely related work in imitation learning, skill extraction, and few-shot generalization.

Imitation Learning: Imitation learning is a supervised learning problem where an agent extracts a policy from a dataset of demonstrations [6, 84]. The two most common approaches to imitation are Behavior Cloning [94, 105] and Inverse Reinforcement Learning (IRL) [80].

BC approaches learn policies $\pi_\theta(a|s)$ that most closely match the state-conditioned action distribution of the demonstration data. IRL approaches learn a reward function from the demonstration data assuming that the demonstrations are near-optimal for the desired task and utilize Reinforcement Learning to produce policies that maximize the reward. For simplicity and to avoid learning a reward function, in this work we aim to learn generalizable skills and use the BC approach. However, two drawbacks of BC are that the imitation policies require a large number of demonstrations and are prone to drifting away from the demonstration distribution during evaluation due to error accumulation [**ross11drift**]. For this reason, BC policies work best when the time horizon of the task is short.

Skill Extraction with Behavioral Priors: Hard-coding prior knowledge into a policy or dynamics model has been considered as a solution to more sample efficient learning, especially in the context of imitation learning [13, 4, 119]. For example, [119] utilizes dynamic movement primitives instead of the raw action space to simplify the learning problem and improve the performance of evolutionary policy search methods. While imposing the structural prior on policy or dynamic models can indeed improve few-shot learning, our method is complementary to these works and proposes to learn behavioral priors from a play dataset. Methods that leverage behavioral priors utilize offline datasets of demonstrations to bias a policy towards the most likely skills in the datasets. While related closely to imitation learning, behavioral priors have been mostly applied to improve Reinforcement Learning. Behavioral priors learned through maximum likelihood latent variable models have been used for structured exploration in RL [114], to solve complex long-horizon tasks from sparse rewards [90], and regularize offline RL policies [135, 87, 77]. While impressive, RL with data-driven behavioral priors does not generalize to new tasks efficiently, often requiring millions of environment interactions to converge to an optimal policy for a new task.

Few-Shot Learning: Few-shot learning [129] has been studied in the context of image recognition [126, 62], reinforcement learning [22], and imitation learning [21]. In the context of reinforcement and imitation learning, few-shot learning is often cast as a meta-learning problem [27, 22, 21], with often specialized datasets of demonstrations that are labeled by tasks. However, there are other means of attaining few-shot generalization and adaptation that do not require such expensive sources of data. For instance, [16] and [86] use Bayesian Optimization for transfer learning to a possibly damaged robot in the real world via learned priors based on big repertoires of controllers in different settings from simulation. However, our problem of interest is skill adaptation that requires no further interaction with the downstream environment during few-shot imitation.

Recently, advances in unsupervised representation learning in natural language processing [97, 10] and vision [48, 14] have shown how a network pre-trained with a self-supervised objective can be finetuned or adjusted with a linear probe to generalize in few-shot or even zero-shot [99] to a downstream task. Our approach to few-shot imitation learning is loosely inspired by the generalization capabilities of networks pre-trained with unsupervised objectives. Our approach first fits a behavioral prior onto an *unlabeled* offline dataset of demonstrations to extract skills and then fits an imitation learning policy over the previously acquired skills to generalize in few-shot to new tasks.

7.3 Approach

Problem Formulation

Few-shot Imitation Learning: We denote a demonstration as a sequence of states and actions: $\tau = \{s_1, a_1, s_2, a_2, \ldots, s_T, a_T\}$. In a few-shot setting we assume access to a small dataset of M such expert demonstrations $\mathcal{D}^{\text{demo}} = \{\tau_i\}_{i=1}^{i=M}$ that fulfill a specific long horizon task in the environment. For instance a sequence of sub-tasks in a kitchen environment such as moving the kettle, turning on the burner, and opening a cabinet door. The goal is to imitate this behavior using only a few available example trajectories.

Skill Extraction: In this work we assume access to an unlabeled offline dataset of prior agent interactions with the environment in the form of N reward-free trajectories $\{\tau_i = \{(s_t, a_t)\}_{t=1}^{t=T_i}\}_{i=1}^{i=N}$. We further assume that these trajectories include semantically meaningful skills that are composable to execute long horizon tasks in the environment. This data can be collected from past tasks that have been attempted or be provided by human experts through teleoperation [137].

Skill extraction refers to an unsupervised learning approach that utilizes this reward-free and task-agnostic dataset to learn a skill policy in form of $\pi_\theta(a|s, z)$ where a is action, s is the current state, and z is the skill. We hypothesize that by combining these skill primitives we can solve semantically similar long-horizon tasks that have not directly been seen during the training. In this work, we propose a new architecture for skill extraction based on continuous latent variable models that enables a semi-parametric evaluation procedure for few-shot imitation learning.

Hierarchical Few-Shot Imitation with Skill Transition Models

Our method, shown in Fig. 7.2, has three components: (i) Skill extraction, (ii) Skill adaptation via fine-tuning on few-shot data, and (iii) Evaluating the skills using a semi-parametric approach to enable few-shot imitation.

(i) Skill Extraction from Offline Data: We define a continuous skill $z_i \in \mathcal{Z}$ as an embedding for a sequence of state-action pairs $\{s_t, a_t, \ldots, s_{t+H-1}, a_{t+H-1}\}$ with a fixed length H. This temporal abstraction of skills has proven to be useful in prior work [90, 1], by allowing a hierarchical decomposition of skills to achieve long horizon downstream tasks. To learn the latent space $\mathcal{Z}$ we propose training a continuous latent variable model with the encoder as $q_\phi(z|s_t, a_t, \ldots, s_{t+H-1}, a_{t+H-1})$ and the decoder as $\pi_\theta(a|s, z)$. The encoder outputs a distribution over the latent variable z that best explains the variation in the state-action pairs in the sub-trajectory.

The encoder is an LSTM that takes in the sub-trajectory of length H and outputs the parameters of a Gaussian distribution as the variational approximation over the true posterior $p(z|s_t, a_t, \ldots, s_{t+H-1}, a_{t+H-1})$. The decoder is a policy that maximizes the log-likelihood of actions of the sub-trajectory conditioned on the current state and the skill. We implement the decoder as a feed-forward network that takes in the current state s_t and the latent vector

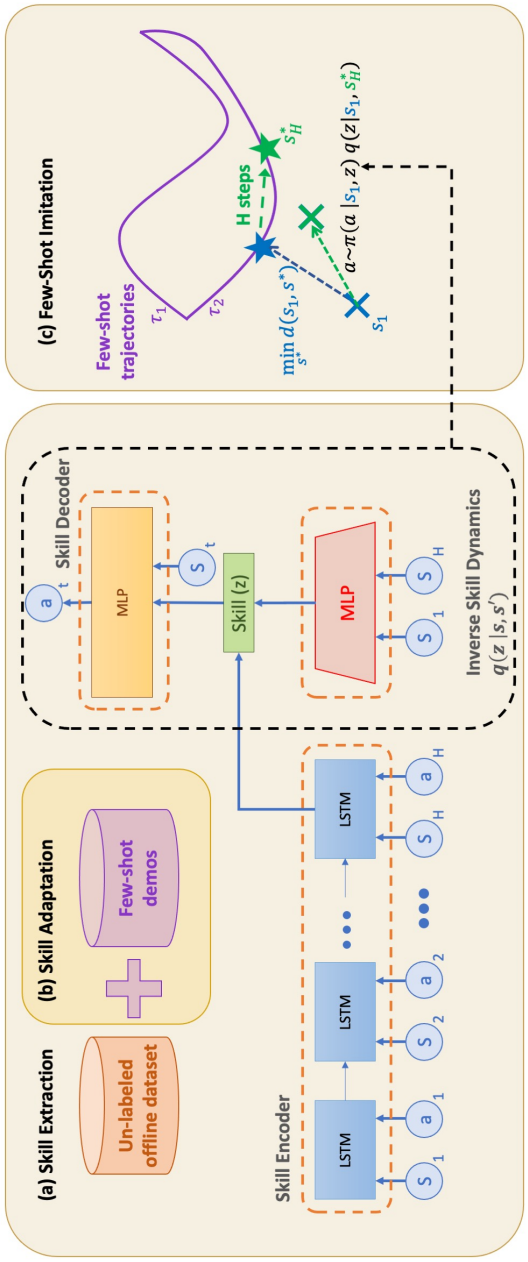


Figure 7.2: Our algorithm – Few-Shot Imitation Learning with Skill Transition Models (FIST) – is composed of three parts: (a) *Skill Extraction:* we fit a skill encoder, decoder, inverse skill dynamics model, and a distance function to the offline dataset; (b) *Skill Adaptation:* For downstream task, we are given a few demonstrations and adapt the skills learned in (a), by fine-tuning the encoder, decoder, and the inverse model. (c) *Few-Shot Imitation:* finally, to imitate the downstream demonstrations, we utilize the distance function to perform a look ahead along the demonstration to condition the inverse model and decode an action.

z and regresses the action vector directly. This architecture resembles prior works on skill extraction [90].

To learn parameters ϕ and θ, we randomly sample batches of H-step continuous sub-trajectories from the training data $\mathcal{D}$ and maximize the evidence lower bound (ELBO):

$$\log p(a_t|s_t) \geq \mathbb{E}_{\tau\sim\mathcal{D}, z\sim q_\phi(z|\tau)}[\underbrace{\log \pi_\theta(a_t|s_t, z)}_{\mathcal{L}_{\text{rec}}} + \beta \underbrace{(\log p(z) - \log q_\phi(z|\tau))}_{\mathcal{L}_{\text{reg}}}] \tag{7.1}$$

where the posterior $q_\phi(z|\tau)$ is regularized by its Kullback-Leibler (KL) divergence from a unit Gaussian prior $p(z) = \mathcal{N}(0, I)$ and β is a parameter that tunes the regularization term [50].

To enable quick few-shot adaptation over skills we learn an inverse skill dynamics model $q_\psi(z|s_t, s_{t+H-1})$ that infers which skills should be used given the current state and a future state that is H steps away. To train the inverse skill dynamics model we minimize the KL divergence between the approximated skill posterior $q_\phi(z|\tau)$ and the output of the state conditioned skill prior. This will result in minimizing the following loss with respect to the parameters ψ:

$$\mathcal{L}_{\text{prior}}(\psi) = \mathbb{E}_{\tau\sim\mathcal{D}}\left[D_{KL}(q_\phi(z|\tau), q_\psi(z|s_t, s_{t+H-1}))\right]. \tag{7.2}$$

We use a reverse KL divergence to ensure that our inverse dynamics model has a broader distribution than the approximate posterior to ensure mode coverage [7]. In our implementation, we use a feed-forward network that takes in the concatenation of the current and future states and outputs the parameters of a Gaussian distribution over z. Conditioning on the future enables us to make a more informed decision on what skills to execute which is a key enabler to few-shot imitation. We jointly optimize the skill extraction and inverse model with the following loss:

$$\mathcal{L}(\phi, \theta, \psi) = \mathcal{L}_{\text{rec}}(\phi, \theta) + \beta\mathcal{L}_{\text{reg}}(\phi) + \mathcal{L}_{\text{prior}}(\psi) \tag{7.3}$$

(ii) Skill Adaption via Fine-tuning on Downstream Data: To improve the consistency between the unseen downstream demonstrations and the prior over skills, we use the demonstrations to fine-tune the parameters of the architecture by taking gradient steps over the loss in Equation 7.3. In the experiments, we ablate the performance of FIST with and without fine-tuning to highlight the differences.

(iii) Semi-parametric Evaluation for Few-shot Imitation Learning: To run the agent, we need to first sample a skill $z \sim q_\psi(z|s_t, s^*_{t+H})$ based on the current state and the future state that it seeks to reach. Then, we can use the low-level decoder $\pi(a_t|z, s_t)$ to convert that sampled skill z and the current state s_t to the corresponding action a_t.

During evaluation, we use the demonstrations $\mathcal{D}^{\text{demo}}$ to decide which state to use as the future state to condition on. For this purpose, we use a learned distance function $d(s, s')$ to measure the distance between the current state s_t and every other state in the demonstrated

trajectories. Then, from the few-shot data we find the closest state s_t^* to the current state according to the distance metric:

$$s_t^* = \min_{s_{ij} \in \mathcal{D}^{\text{demo}}} d(s_t, s_{ij}) \tag{7.4}$$

where s_{ij} is the j^{th} state in the i^{th} trajectory in $\mathcal{D}^{\text{demo}}$. We then condition the inverse dynamics model on the current state s_t and the state s_{t+H}^*, H steps ahead of s_t^*, within the trajectory that s_t^* belongs to. If by adding H steps we reach the end of the trajectory, we use the end state within the trajectory as the target future state. The reason for this look-ahead adjustment is to ensure that the sampled skill always makes progress towards the future states of the demonstration. After the execution of action a_t according to the low-level decoder, the process is repeated until the fulfillment of the task. The procedure is summarized in Algorithm 4.

Algorithm 4 FIST: Evaluation Algorithm

1: **Inputs:** Fine-tuned inverse skill dynamics model $q_\psi(z|s_t, s_{t+H-1})$, fine-tuned skill policy $\pi_\theta(a|s, z)$, learned distance function $d(s, s')$, downstream demonstration $\mathcal{D}^{\text{demo}}$
2: Initialize the environment to s_0
3: **for** each $t = [1 \dots T]$ **do**
4: Pick $s_t^{*'} = \text{LookAhead}(\min_{s \in \mathcal{D}^{\text{demo}}} d(s_t, s))$
5: Sample skill $z \sim q_\psi(z|s_t, s_t^{*'})$
6: Sample action $a \sim \pi_\theta(a|s_t, z)$
7: $s_t \leftarrow \text{env.step}(a)$

Learning the distance function

To enable few-shot imitation learning we use a learned distance function to search for a goal state that is "close" to the current state of the agent. Learning contrastive distance metrics has been successfully applied in prior work both in navigation and manipulation experiments. Both [72] and [23] utilize contrastive distance functions to build semi-parametric topological graphs, and [108] and [110] use a similar approach for visual navigation. Inspired by the same idea, we also use a contrastive loss such that states that are H steps in the future are close to the current state while all other states are far.

We wish to learn an encoding such that our distance metric d is the euclidean distance between the encoded states.

$$d(s, s') = ||h(s) - h(s')||^2 \tag{7.5}$$

To learn the encoder h, we optimize a contrastive loss on encodings of the current and future states along the same trajectory. At each update step we have a batch of trajectories

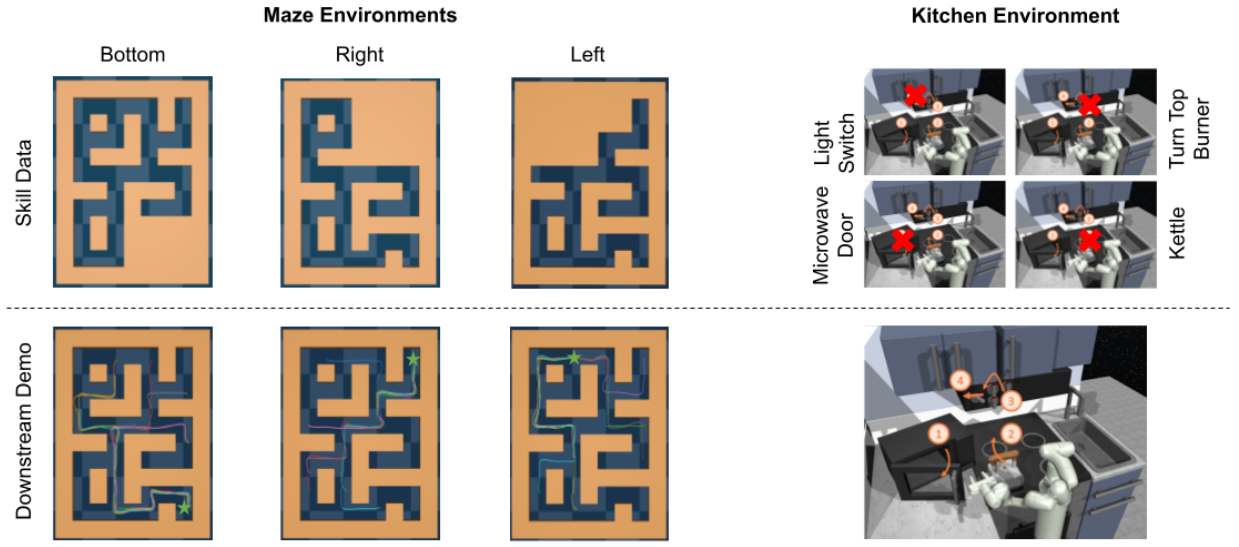


Figure 7.3: **Top**: In each environment, we block some part of the environment and collect task-agnostic and reward-free trajectories for extracting skills. In the kitchen environment, red markers indicate the objects that are excluded. **Bottom**: For downstream demonstrations, we use 10 expert trajectories that involve unseen parts of the maze or manipulation of unseen objects.

$\tau_1, ..., \tau_B$ where $\tau_i = s_{i,1}, ..., s_{i,H}$. For $s_{i,1}$ the positive sample is $s_{i,H}$ and the negative samples are $s_{j,H}$ where $j \neq i$. We use the InfoNCE Loss [83],

$$\mathcal{L}_q = -\log \frac{\exp(q^T W k)}{\sum_{i=0}^{K} \exp(q^T W k_i)}, \tag{7.6}$$

with query $q = h(s_t)$ as the encoded starting state, and the keys $k = h(s_{t+H})$ as the encoded future states along the K trajectories in the dataset $\mathcal{D}$. By doing so, we are attracting the states that are reachable after H steps and pushing those not reachable far out. This approach has been used successfully in prior work in [72, 23] and we use this method for a generalizable solution for finding the state to condition the future on. This particular choice of distance function is a means to an end for obtaining the future state to condition on and further research is required for investigating similar choices for a robust performance across a diverse set of tasks. We leave this investigation for future work.

7.4 Experiments

In the experiments we are interested in answering the following questions: (i) Can our method successfully imitate unseen long-horizon downstream demonstrations? (ii) What is the importance of the semi-parametric approach vs. future conditioning? (iii) Is pre-training and fine-tuning the skill embedding model necessary for achieving a high success rate?

Pre-training

The training for both skill extraction and fine-tuning were done on a single NVIDIA 2080Ti GPU. Skill extraction takes approximately 3-4 hours, and fine-tuning requires less than 10 minutes. Our codebase builds upon the SPiRL released code and is located at `https://github.com/kouroshhakha/fist`. Hyperparameters used for training are listed in Table 7.1.

Fine-tuning

Fine-tuning for both FIST and SPiRL follows the same implementation details. It is done only on $\mathcal{D}^{\text{demo}}$ which includes 10 trajectories of the agent fulfilling the long horizon task. These trajectories are segmented into sub-trajectories of length $H = 10$ first (similar to pre-training), and then we update all the network parameters[1] by minimizing the loss in equation 7.3, for 50 epochs of the small dataset. The original training was done on 200 epochs of the large and diverse dataset. Everything else, including batch size, learning rate, and the optimizer remains the same between pre-training and fine-tuning. The hyper-parameters for fine-tuning are listed at the bottom of Table 7.1.

[1]During our initial experimental analysis, we tried a version of FIST that only finetuned the inverse skill dynamics model and that did not show any meaningful results. We hypothesize that without fine-tuning the VAE components, the skill-set of the agent will not include the out of distribution parts of the task.

Table 7.1: Training Hyperparameters

Hyperparameter	Value
Contrastive Distance Metric	
Encoder output dim	32
Encoder Hidden Layers	128
Encoder # Hidden Layers	2
Optimizer	Adam($\beta_1 = 0.9$, $\beta_2 = 0.999$, LR=1e-3)
Skill extraction	
Epochs	200
Batch size	128
Optimizer	Adam($\beta_1 = 0.9$, $\beta_2 = 0.999$, LR=1e-3)
H (sub-trajectory length)	10
β	5e-4 (Kitchen), 1e-2 (Maze)
Skill Encoder	
dim-$\mathcal{Z}$ in VAE	128
hidden dim	128
# LSTM Layers	1
Skill Decoder	
hidden dim	128
# hidden layers	5
Inverse Skill Dynamic Model	
hidden dim	128
# hidden layers	5
Fine-tuning	
Epochs	50
Epoch cycle train	10
Batch size	128
Optimizer	Adam($\beta_1 = 0.9$, $\beta_2 = 0.999$, LR=1e-3)

Environments

We evaluate the performance of FIST on two simulated navigation environments and a robotic manipulation task from the D4RL benchmark as shown in Figure 7.3. To ensure generalizability to out-of-distribution tasks we remove some categories of trajectories from the offline data and at test time, we see if the agent can generalize to those unseen trajectories.

Specifically for Pointmass and Ant environments we block some regions of the maze and at test time, we provide demonstrations that navigate into the excluded region; and for Kitchen, we exclude interactions with a few selected objects and at test time, we provide demonstrations that involve manipulating the excluded object.

In this section, we explain the tasks and the procedure for data collection for both pre-training the skills and the downstream fine-tuning in each environment separately. The instruction for downloading the dataset as well as its generation is included in our code repository.

PointMaze. In this environment, the task is to navigate a point mass through a maze, from a start to a goal location. The outline of the maze is shown in Figure 7.3. We train the skills on three different datasets, each blocking one side of the maze. To test the method's ability to generalize to unseen long-horizon tasks, we use 10 expert demonstrations that start from random places in the maze, but end at a goal within the blocked region. This ensures that our demonstrated trajectories are out of distribution compared to training data. We evaluate the performance by measuring the episode length and the success rate in reaching the demonstrated goals. The data for PointMaze is collected using the same scripts provided in the D4RL dataset repository [29]. To generate the pre-training dataset we modified the maze outline to have parts of it blocked. Then we run the same oracle way-point controller on the new maze map and collect continuous trajectories of the agent navigating to random goals, for a total of 4 million transitions. For downstream demonstrations we unblock the blocked region, pick a fixed goal within that region, and have the agent navigate from random reset points to the picked goal location. The observation consists of the (x, y) location and velocities.

AntMaze. The task is to control a quadruped ant to run to different parts of the maze. The layout of the maze is the same as PointMaze, and the same sides are blocked off. Similar to PointMaze we measure the episode length and success rate as our evaluation metric. The data for AntMaze is solely based on the "ant-large-diverse-v0" dataset in D4RL. To construct the pre-training data we filter out sub-trajectories that contain states within the blocked region. By doing so, we effectively exclude the region of interest from the pre-training dataset. The result is datasets with 47165, 58329, and 50237 transitions, for the Bottom, Right, and Left blocked regions respectively. For the downstream demonstrations, we randomly select 10 from the excluded trajectories that start outside the blocked region and end in the blocked region, shown in Figure 7.3.

Kitchen. The task is to use a 7-DoF robotic arm to manipulate different parts of a kitchen environment in a specific order (e.g. open a microwave door or move the kettle)[2]. During skill extraction, we pre-process the offline data to exclude interactions with certain objects in the environment (e.g. we exclude interactions with the kettle). However, for the demonstrations, we pick four sub-tasks one of which includes the objects that were excluded from the skill dataset (e.g. if the kettle was excluded, we pick the task to be to open the

[2]The environment dynamics are still stochastic and therefore, a simple replication of actions would not fulfill the tasks robustly

microwave, move the kettle, turn the top burner, and slide the cabinet door). In evaluation, for completion of each sub-task in the order consistent with the downstream demonstrations, the agent is awarded a reward of 1.0 for a total max reward of 4.0 per episode.

The pool of demonstrations is downloaded from the repository of [37] located at `https://github.com/google-research/relay-policy-learning`. There are a total of 24 multi-task long horizon sets of trajectories that the data is collected from. Each trajectory set is sampled at least 10 times via VR teleoperation procedure and the filenames indicate what the agent is trying to achieve (e.g. microwave-kettle-switch-slide). For creating the pre-training data, we filter the trajectory sets, solely based on the filenames that do not include the keyword for the task of interest. This will essentially remove all multi-task trajectories that include the sub-task and not just the part that we do not want. It is also worth noting that the excluded object (e.g. the kettle) is still part of the environment and all its state vectors are still visible to the agent, despite the exclusion of the "interaction" with that object. The dataset size for each case is shown in Table 7.2.

Table 7.2: Kitchen pre-training dataset sizes

Tasks	**# of trajectories**
Microwave, Kettle, **Top Burner**, Light Switch	285
Microwave, Bottom Burner, Light Switch, Slide Cabinet	236
Microwave, **Kettle**, Slide Cabinet, Hinge Cabinet	230
Microwave, Kettle, **Slide Cabinet**, Hinge Cabinet	198

Results

We use the following approaches for comparison:

- **BC+FT**: Trains a behavioral cloning agent (i.e. $\pi_\theta(a|s)$) on the offline dataset $\mathcal{D}$ and fine-tunes to the downstream dataset $\mathcal{D}^{\text{demo}}$.
- **SPiRL**: This is an extension of the existing skill extraction methods to imitation learning over skill space [90]. The skill extraction method in SPiRL [90] is very similar to FIST, but instead of conditioning the skill prior on the future state it only uses the current state. To adapt SPiRL to imitation learning, after pre-training the skills module, we fine-tune it on the downstream demonstrations $\mathcal{D}^{\text{demo}}$ (instead of finetuning with RL as proposed in the original paper). After fine-tuning we execute the skill prior for execution.
- **FIST (ours)**: This runs our semi-parametric approach after learning the future conditioned skill prior. After extracting skills from $\mathcal{D}$ we fine-tune the parameters on

the downstream demonstrations $\mathcal{D}^{\text{demo}}$ and perform the proposed semi-parametric approach for evaluation.

Figure 7.4 compares the normalized average score on each unseen task from each domain. Each tick on x-axis presents either an excluded region (for Maze) or an excluded object (for Kitchen). The un-normalized scores are included in tables 7.3 and 7.4.

Here, we provide a summary of our key findings:

1. In the PointMaze environment, FIST consistently succeeds in navigating the point mass into all three goal locations. The skills learned by SPiRL fail to generalize when the point mass falls outside of the training distribution, causing it to get stuck in corners. While BC+FT also solves the task frequently in the Left and Bottom goal locations, the motion of the point mass is sub-optimal, resulting in longer episode lengths.

2. In the AntMaze environment, FIST achieves the best performance compared to the baselines. SPiRL and BC+FT make no progress in navigating the agent towards the goal while FIST can frequently reach the goals in the demonstrated trajectories. We believe that the low success rate numbers in this experiment are due to the low quality of trajectories that exist in the offline skill dataset $\mathcal{D}$. In the dataset, we see many episodes with the ant falling over, and FIST's failure cases also demonstrate the same behavior, hence resulting in a low success rate. We hypothesize that with a better dataset FIST can achieve a higher success rate number.

3. In the kitchen environment, FIST significantly outperforms SPiRL and BC+FT. FIST can successfully complete **3 out of 4** long-horizon object manipulation tasks in the same order required by the task. In one of these long-horizon tasks, all algorithms perform similarly poor.

We believe that such behavior is because the majority of the trajectories in the pre-training dataset start with a microwave task and removing this sub-task can substantially decrease the diversity and number of trajectories seen during pre-training.

Ablation Studies

In this section, we study different components of the FIST algorithm to provide insight into the contribution of each part.

Effect of the Semi-parametric evaluation and future conditioning of the skill prior

We recognize two major differences between FIST and SPiRL: (1) future conditioning of the skill prior during skill extraction and (2) the semi-parametric approach for picking the future state to condition on. To separately investigate the contribution of each part we performed

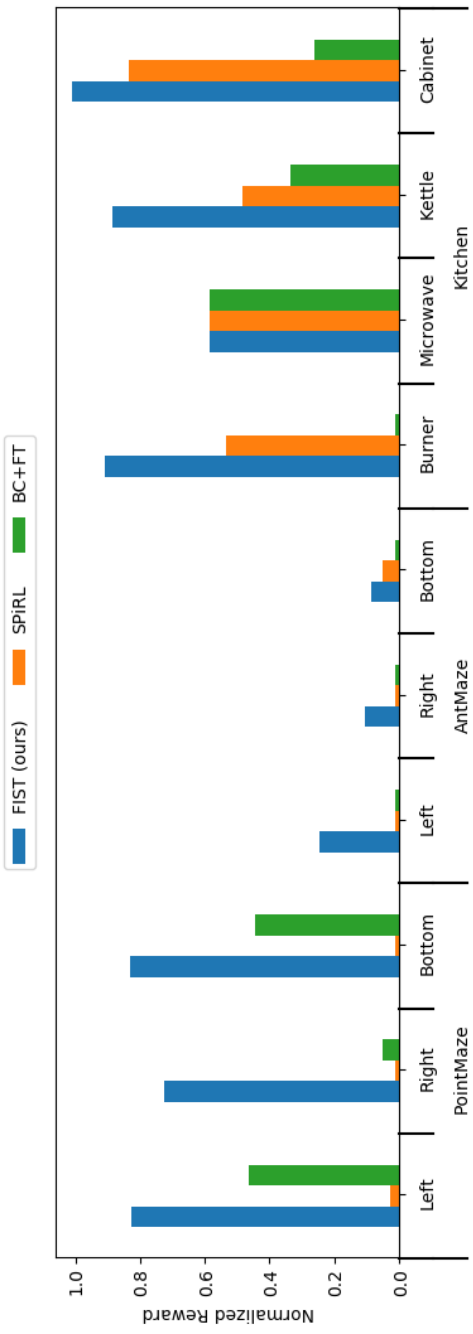


Figure 7.4: Normalized Reward on all of our environments, and their excluded regions / objects. For maze, the episode length is subtracted from the maximum episode length, which is then divided by the maximum episode length. For kitchen, the reward is the number of sub-tasks completed in order, and normalized by the maximum of 4.0

Table 7.3: Comparison of our approach to other baselines on the Maze environments. For each experiment we report the average episode length from 10 fixed starting positions with the standard error across 10 evaluation runs (*lower* is better). We also report success rate and its standard deviation. The maximum episode length for PointMaze and AntMaze are 2000 and 1000, respectively.

		FIST (Ours)		**SPiRL**		**BC+FT**	
Blocked Region	Environment	Episode Length	Success Rate	Episode Length	Success Rate	Episode Length	Success Rate
Left	PointMaze	**329.09 ± 5.62**	1.0 ± 0.00	1966.7 ± 32.54	0.02 ± 0.04	1089.76 ± 173.74	0.74 ± 0.11
Right	PointMaze	**440.47 ± 50.81**	0.99 ± 0.03	2000 ± 0	0.0 ± 0.0	1918.99 ± 43.65	0.07 ± 0.06
Bottom	PointMaze	**491.67 ± 78.44**	0.91 ± 0.05	2000 ± 0	0.0 ± 0.0	1127.47 ± 148.24	0.87 ± 0.10
Left	AntMaze	**764.36 ± 8.93**	0.32 ± 0.04	1000 ± 0	0.0 ± 0.0	1000 ± 0	0.0 ± 0.0
Right	AntMaze	**903.98 ± 12.01**	0.22 ± 0.12	1000 ± 0	0.0 ± 0.0	1000 ± 0	0.0 ± 0.0
Bottom	AntMaze	**923.22 ± 6.36**	0.21 ± 0.07	957.85 ± 8.62	0.12 ± 0.07	1000 ± 0	0.0 ± 0.0

Table 7.4: Comparison of average episode reward for our approach against other baselines on the KitchenRobot environment. The average episode reward (with a max. of 4) along with its standard error is measured across 10 evaluation runs (*higher* is better). Each bolded keyword indicates the task that was excluded during skill data collection.

Task (Unseen)	**Environment**	**FIST (Ours)**	**SPiRL**	**BC+FT**
Microwave, Kettle, **Top Burner**, Light Switch	KitchenRobot	$\mathbf{3.6 \pm 0.16}$	2.1 ± 0.48	0.0 ± 0.0
Microwave, Bottom Burner, Light Switch, Slide Cabinet	KitchenRobot	$\mathbf{2.3 \pm 0.5}$	$\mathbf{2.3 \pm 0.5}$	$\mathbf{2.2 \pm 0.28}$
Microwave, **Kettle**, Slide Cabinet, Hinge Cabinet	KitchenRobot	$\mathbf{3.5 \pm 0.3}$	1.9 ± 0.09	1.3 ± 0.47
Microwave, Kettle, **Slide Cabinet**, Hinge Cabinet	KitchenRobot	$\mathbf{4.0 \pm 0.0}$	3.3 ± 0.38	1.0 ± 0.32

an ablation with the following modifications of FIST and SPiRL on the kitchen environment: **FIST (Euc.)** is similar to our FIST experiment, except that we use the Euclidean distance on the raw state space, rather than our learned contrastive distance for lookup. Comparison to this baseline measure the importance of our contrastive distance method, compared to a simpler alternative; **SPiRL (closest)** uses the contrastive distance function to look-up the closest state in the demonstrations to sample the skill via $p(z|s_{\text{closest}})$; and **SPiRL (H-step)** uses the same H-step-ahead look-up approach as FIST and samples a skill based on $p(z|s_{\text{closest}+H})$.

Table 7.5: Ablation on the impact of semi-parametric approach and the future conditioning of skill prior

Task (Unseen)	FIST (ours)	FIST (Euc.)	SPiRL	SPiRL (closest)	SpiRL (H-steps)
Microwave, Kettle, **Top Burner**, Light Switch	$\mathbf{3.6 \pm 0.16}$	3.3 ± 0.15	2.1 ± 0.48	0.0 ± 0.0	0.2 ± 0.13
Microwave, Bottom Burner, Light Switch, Slide Cabinet	2.3 ± 0.5	$\mathbf{2.8 \pm 0.42}$	2.3 ± 0.49	2.8 ± 0.44	0.0 ± 0.0
Microwave, **Kettle**, Slide Cabinet, Hinge Cabinet	$\mathbf{3.5 \pm 0.3}$	3.0 ± 0.32	1.9 ± 0.29	0.0 ± 0.0	0.0 ± 0.0
Microwave, Kettle, **Slide Cabinet**, Hinge Cabinet	$\mathbf{4.0 \pm 0.0}$	1.3 ± 0.14	3.3 ± 0.38	2.4 ± 0.51	1.5 ± 0.41

A comparison of SPiRL results suggests that the performance would degrade if the semi-parametric lookup method is used to pick the conditioning state in SPiRL. This is not surprising since the model in SPiRL is trained to condition on the current state and is unable to pick a good skill z based on $p(z|s_{\text{closest}})$ in SPiRL (closest) or $p(z|s_{\text{closest}+H})$ in SPiRL (H-steps). Therefore, it is crucial to have the prior as $p(z|s_t, s_{t+H})$, so that we can condition on both the current and future goal state.

We also note that FIST (Euc.) results are slightly worse in 3 out of 4 cases. This suggests that (1) contrastive distance is better than Euclidean to some degree (2) the environment's state space is simple enough that Euclidean distance still works to a certain extent. It is worth noting that the Euclidean distance, even with comparable results to the contrastive distance in the kitchen environment, is not a general distance metric to use and would quickly lose advantage when state representation gets complex (e.g. pixel representation).

The effect of skill pre-training and fine-tuning on FIST

To adjust the pre-trained skill-set to OOD tasks (e.g. moving the kettle while it is excluded from the skill dataset) FIST requires fine-tuning on the downstream demonstrations. We hypothesize that without fine-tuning, the agent should be able to perfectly imitate the demonstrated sub-trajectories that it has seen during training, but should start drifting away when encountered with an OOD skill. We also hypothesize that pre-training on a large dataset, even if it does not include the downstream demonstration sub-trajectories, is crucial for better generalization. Intuitively, pre-training provides a behavioral prior that is easier to adapt to unseen tasks than random initialization.

To examine the impact of fine-tuning, we compare FIST with *FIST-no-FT* which directly evaluates the semi-parametric approach with the model parameters trained on the skill dataset without fine-tuning on the downstream trajectories. To understand the effect of pre-training, we compare FIST with *FIST-no-pretrain* which is not pre-trained on the skill dataset. Instead, we directly train the latent variable and inverse skill dynamics model on the downstream data and perform the semi-parametric evaluation of the FIST algorithm.

From the results in Table 7.6, we observe that fine-tuning is a critical component for adapting to OOD tasks. The scores on *FIST-no-FT* indicate that the agent is capable of fulfilling the sub-tasks seen during skill pre-training but cannot progress onto unseen tasks without fine-tuning. Based on the scores on *FIST-no-pretrain*, we also find that the pre-training on a rich dataset, even when the downstream task is directly excluded, provides sufficient prior knowledge about the dynamics of the environment and can immensely help with generalization to unseen tasks via fine-tuning.

Imitation Learning over skills vs. atomic actions

FIST is comprised of two coupled pieces that are both critical for robust performance: the inverse dynamics model over skills and the non-parametric evaluation algorithm. In this experiment we measure the influence of inverse skill dynamics model $q_\psi(z|s_t, s_{t+H-1})$.

Table 7.6: We ablate the use of pre-training on offline data, as well as fine-tuning on downstream demonstrations. FIST-no-FT removes the fine-tuning on the downstream demonstration step in FIST, while FIST-no-pretrain trains the skills purely from the given downstream data. Without seeing the subtask, FIST-no-FT is unable to solve the downstream subtask. Trained on only downstream data, FIST-no-pretrain is unable to properly manipulate the robot.

Task (Unseen)	FIST (ours)	FIST-no-FT	FIST-no-pretrain
Microwave, Kettle, **Top Burner**, Light Switch	$\mathbf{3.6 \pm 0.16}$	2.0 ± 0.0	0.5 ± 0.16
Microwave, Bottom Burner, Light Switch, Slide Cabinet	$\mathbf{2.3 \pm 0.5}$	0.0 ± 0.0	0.7 ± 0.15
Microwave, **Kettle**, Slide Cabinet, Hinge Cabinet	$\mathbf{3.5 \pm 0.3}$	1.0 ± 0.0	0.0 ± 0.0
Microwave, Kettle, **Slide Cabinet**, Hinge Cabinet	$\mathbf{4.0 \pm 0.0}$	2.0 ± 0.0	0.8 ± 0.13

An alternative baseline to learning the skill dynamics model is to learn an inverse dynamics model on atomic actions $q_\psi(a_t|s_t, s_{t+H-1})$ and perform goal-conditioned behavioral cloning (Goal-BC). This model outputs the first action a_t required for transitioning from s_t to s_{t+H-1} over H steps. We can combine this model with FIST's non-parametric module to determine the s_{t+H-1} to condition on during the evaluation of the policy. As shown in Table 7.7, temporal abstraction obtained in learning an inverse skill dynamics model is a critical factor in the performance of FIST.

Table 7.7: We ablate the use of our inverse skill dynamics model by replacing it with an inverse dynamics model on atomic actions. The baseline ablations only succeed on one out of the four tasks. BC learns an inverse dynamics model that takes in the state as input and outputs a distribution over atomic actions. Goal-BC uses both state and the goal (sub-task) as input.

Task (Unseen)	FIST (ours)	Goal-BC
Microwave, Kettle, **Top Burner**, Light Switch	$\mathbf{3.6 \pm 0.16}$	0.0 ± 0.0
Microwave, Bottom Burner, Light Switch, Slide Cabinet	$\mathbf{2.3 \pm 0.5}$	1.2 ± 0.3
Microwave, **Kettle**, Slide Cabinet, Hinge Cabinet	$\mathbf{3.5 \pm 0.3}$	1.8 ± 0.44
Microwave, Kettle, **Slide Cabinet**, Hinge Cabinet	$\mathbf{4.0 \pm 0.0}$	0.9 ± 0.1

Is our contrastive distance function an optimal approach for picking the future state to condition on?

For environments such as PointMaze, where we have access to the same waypoint controller that generates the demonstrations, we can use the ground truth environment dynamics to calculate the oracle state that the waypoint controller would be at, H steps in the future, and use that as the goal. This waypoint controller is not available for the kitchen environment, since the demonstrations were collected using VR teleoperation.

In Table 7.8, FIST (Oracle) uses the waypoint controller oracle look-up. The results show that with better future conditioning, an oracle approach can solve the pointmaze task even better than FIST with the contrastive distance. Improving the current semi-parametric approach for obtaining the future conditioning state could be a very interesting direction for future work.

Table 7.8: We ablate FIST against an oracle version on pointmaze which uses the ground truth way point planner that has access to the exact state that the agent will end up at H-steps in the future (if it commits to the optimal path).

Section	FIST		FIST (oracle)	
	Episode Length	Success Rate	Episode Length	Success Rate
Left	329.09 ± 5.62	1.0 ± 0.00	236.00 ± 1.02	1.0 ± 0.00
Right	440.47 ± 50.81	0.99 ± 0.03	280.93 ± 5.61	1.0 ± 0.00
Bottom	491.67 ± 78.44	0.91 ± 0.05	269.89 ± 3.75	1.0 ± 0.00

Skill re-sampling frequency Although the skills are conditioned on states H steps into the future, we found executing the skill for one step, and then re-sampling the skill latent to be more efficient. In Table 7.9, we quantitatively compare the difference in episode length for the PointMaze environment using different re-sampling frequencies. As the agent progresses through the environment, the optimal state s_{t+H-1} changes, and so does the optimal skill z. This difference is shown in Table 7.9, where sampling less often results in less optimal trajectories and longer episode lengths. However, planning every H step is still able to solve the task, and can be used if calculating the goal state s_{t+H-1} using the semi-parametric model is expensive.

Table 7.9: Different skill re-sampling schemes. Shown are $t = 1$, which are the results in Table 7.3. We compare the results to when $t = 5$, and when the skill is resampled after H steps, $t = 10$.

	$t = 1$		$t = 5$		$t = 10$	
Blocked Region	Episode Length	Success Rate	Episode Length	Success Rate	Episode Length	Success Rate
Left	329.09 ± 5.62	1.0 ± 0.00	324.55 ± 6.09	1.0 ± 0.00	338.04 ± 9.80	1.0 ± 0.00
Right	440.47 ± 50.81	0.99 ± 0.03	474.26 ± 69.45	0.98 ± 0.04	465.05 ± 52.38	1.0 ± 0.00
Bottom	491.67 ± 78.44	0.91 ± 0.05	510.45 ± 3.58	0.9 ± 0.00	520.03 ± 47.91	0.89 ± 0.03

One-shot Imitation Learning

The FIST algorithm can be directly evaluated on one-shot in-distribution downstream tasks without any fine-tuning. In this experiment, we want to see if the agent can pick up the right mode within its skill-set with only one demonstration for fulfilling a long-horizon task in the kitchen environment. The difference between this experiment and our main result is that the downstream task is within the distribution of its pre-trained skill-set. This is still a challenging task since the agent needs to correctly identify the desired mode of skills.

We hypothesize that in SPiRL, the skill prior is only conditioned on the current state and therefore is, by definition, a multi-modal distribution and would require more data to adapt to a specific long-horizon trajectory. For instance, in the kitchen environment, after opening the microwave door, the interaction with any other objects in the environment is a possible choice of skills that can be invoked. However, in FIST, by conditioning the skill prior on the future states, we fit a uni-modal distribution over skills. In principle, there should be no need for fine-tuning for invoking those skills within the distribution of the pre-trained skill set.

We compare our approach to SPiRL (Section 7.4) as a baseline. In addition, we can provide supervision on which skills to invoke to fulfill the long-horizon task by fine-tuning SPiRL (hence *SPiRL-FT*) for a few epochs on the downstream demonstration. As summarized in Table 7.10, FIST, without any fine-tuning, can fulfill all the long-horizon tasks listed with almost no drift from the expert demonstration. We also see that it is tricky to fine-tune SPiRL in a one-shot setting, as fine-tuning only on one demonstration may cause over-fitting and degradation of performance.

Table 7.10: With all subtasks seen in the skill dataset, FIST is able to imitate a long-horizon task in the kitchen environment. We compare to a baseline method, SPiRL, which fails to follow the single demo.

Order of tasks (seen in the skill dataset)	FIST (ours)	SPiRL-FT	SPiRL-no-FT
Kettle, Bottom Burner, Slide Cabinet, Hinge Cabinet	$\mathbf{4.0 \pm 0.0}$	0.8 ± 0.19	2.4 ± 0.35
Kettle, Top Burner, Light Switch, Slide Cabinet	$\mathbf{3.8 \pm 0.19}$	0.5 ± 0.16	1.1 ± 0.22
Microwave, Kettle, Slide Cabinet, Hinge Cabinet	$\mathbf{4.0 \pm 0.0}$	1.1 ± 0.22	1.0 ± 0.37
Top Burner, Bottom Burner, Slide Cabinet, Hinge Cabinet	$\mathbf{4.0 \pm 0.0}$	0.1 ± 0.1	0.6 ± 0.25

7.5 Broader Impacts and Limitations

Limitations As with all imitation learning methods, the performance of FIST is related to the quality of the provided demonstrations. Concretely, when the skill training demonstrations are poor, we expect the extracted skills to be also sub-optimal, thus, hurting downstream imitation performance. To better understand this limitation, we analyze an extremely noisy version of the PointMaze dataset and use it for skill extraction. As shown

Environment	Episode Length	Success Rate
PointMaze	621.02 ± 69.87	1.0 ± 0.0

Table 7.11: We evaluate FIST on the maze environment with goal at the bottom when the inverse skill model is trained on an extremely noisy dataset. In this case, FIST achieves sub-optimal performance, or is unable to imitate the test time demonstration.

in Table 7.11, despite achieving a high success rate, the episode length is substantially worse than FIST trained on expert data.

Learning structured skills from noisy offline data is an exciting direction for future research.

Broader Impacts The ability to extract skills from offline data and adapt them to solve new challenging tasks in few-shot, could be impactful in domains where large offline datasets are available but control is challenging and cannot be manually scripted. Examples of such domains include autonomous vehicle navigation, warehouse robotics, digital assistants, and perhaps in the future, home robots. However, there are also negative potential consequences. First, since in real-world settings offline data will be collected from users at scale there will likely be privacy concerns, especially for video data collected from users' cars or homes. Additionally, since FIST extracts skills, without labeled data, quality for large datasets becomes increasingly opaque and if there are harmful skills or behavior present in the dataset FIST may extract those and use them during deployment which could have unintended consequences. A promising direction for future work is to include a human in the loop for skill verification.

7.6 Conclusion

We present FIST, a semi-parametric algorithm for few-shot imitation learning for long-horizon tasks that are unseen during training. We use previously collected trajectories of the agent interacting with the environment to learn a set of skills along with an inverse dynamics model that is then combined with a non-parametric approach to keep the agent from drifting away from the downstream demonstrations. Our approach can solve long-horizon challenging tasks in the few-shot setting where other methods fail.

Chapter 8

Conclusion

8.1 Thesis Contribution

In this thesis, we extended the idea of pre-training neural networks for sample efficient generalization to two categories of problems: 1) design optimization with a focus on analog-mixed signal circuit design and 2) in control with a focus on few-shot imitation learning.

For solving circuit design optimization problems we began by re-visiting evolutionary algorithms as the optimization backbone of our methods and proposed a modeling idea that results in boosting the sample efficiency of these methods. In Chapter 4, we presented BagNet, an evolutionary black-box optimization algorithm, which replaces the expensive simulation in the loop with a deep neural network that is trained with the online samples that we collect during optimization. We showed that if the neural network is architected as a design discriminator model, it can result in a significant reduction in the number of simulations that are needed to reach the desired specifications using the underlying evolutionary algorithm. We showed that this reduction in sample efficiency enables direct automated layout optimization of designs that are large and complicated even for human designers, paving the way for improving productivity and utility of existing analog design flows.

In Chapter 5, we proposed an idea for pre-training analog circuit models that is based on learning to model the DC behavior. We showed that by pre-training graph neural networks to predict the DC behavior of a variety of circuits across different topologies or even on the same topology we can build in some circuit-specific domain knowledge about the representations of each device that we can fine-tune for more sample efficient circuit modeling or design optimization. We demonstrated this hypothesis, by replacing BagNet's feature extractor with the pre-trained graph neural network backbones and showed that the improvement in the model quality will result in another significant reduction in the number of simulations required for the optimization.

We then revisited the idea of pre-training in other more sample efficient optimization methods like Bayesian optimization and proposed JUMBO (Chapter 6), a multi-task bayesian optimization algorithm that can use *large* auxiliary datasets from the domain, to efficiently

transfer prior knowledge to a new downstream target objective.

In Chapter 7, we turned to the problem of pre-training agents to do sample efficient control. We proposed FIST, a few-shot imitation learning algorithm that uses previously collected trajectories of the agent to extract permissible skills in the environment, and showed that we can efficiently re-use those skills to fulfill unseen downstream tasks with only a few examples from the task.

8.2 Future of Deep Learning in Analog Circuit Design Automation

On circuit design optimization, I believe that up-scaling the pre-training dataset and downstream problems could be an interesting direction for expanding this research. Currently, in our work, the pre-training dataset is narrowly defined around a single topology or a few minor topological variations. It would be interesting to see how the pre-trained models would behave if the pre-training dataset covers more diverse circuit topologies with varying levels of performance or functionality. Reaching this goal requires novelty in data collection methods and how one can use software techniques to efficiently scale up the data generation and simulation processes. For pre-training, we can certainly use fast simulators like `NGSPICE` that preserve the functionality and behavioral information of different topologies to learn re-usable features. Thereby not limiting the pre-training dataset collection procedure by the long run-times of layout simulations.

Analog circuits generally can be categorized into a handful of classes and every new invention of a circuit is usually a realization of such category. To name a few we have operational amplifiers, trans-impedance amplifiers, sense amplifiers, digital to analog converters, analog to digital converters, charge-pump, phase lock loops, .etc. which are different classes of functional blocks in circuit systems. As a general road map for the adoption of these models at scale, it is currently not clear whether having one separate model for each circuit class is going to be better than a single unified model for all circuits. It would be an interesting future work to see which one is better; on one hand, one single unified model can leverage the learning signal encoded in the shared sub-structures of these different classes of circuits to learn better representations, but on the other hand, finding a coherent learning objective for pre-training on all these circuit classes on a single model can be a challenge itself. One idea is to use a single GNN backbone for all circuits but has different prediction heads for each circuit class that back-propagates the learning error for each one separately.

Overall, this idea of pre-training circuit representations from transistor-level graph descriptions seems to be a promising approach, and scaling it up could result in foundational transistor-level models that can be used with little to no fine-tuning for novel downstream tasks including accelerated layout optimization or technology porting which are currently the biggest pain-points and productivity barriers for analog circuit designers.

Such foundational models, in principle, can be used for improving the sample efficiency

of any model-based optimization algorithm including Bayesian optimization. In JUMBO we showed a proof of concept of the effectiveness of pre-training fully-connected architectures. However, this method can further benefit from the pre-trained GNN architectures discussed in Chapter 5. Therefore, a natural future research direction would be to see how we can re-use such general circuit representations in the context of Bayesian optimization to achieve the state-of-the-art sample efficiency for layout optimization.

8.3 Future of Pre-trained Agents for Sample Efficient Control

Pre-training agents for sample efficient control is also a research area that is growing at a significant pace. Especially after the success of recent works on bridging the sample efficiency of image-based RL and state-based RL [117] via using general visual consistency objectives like contrastive loss during RL training. We are also seeing emerging works on pre-training agents for learning visual representations that can be used for control [78, 85] which is another promising factor we should consider when thinking about pre-trained agents. However, being able to precisely control the environment with little to no supervision requires more prior knowledge than just learning the visual representation of the world.

In my opinion, this knowledge should come from two distinct learning paradigms: passive and active pre-training. Our work, FIST, essentially combined the two paradigms in one algorithm and learned the permissible skills in the world that the agent could do via looking at the previously skillful trajectories from the same agent in the environment. To enable more generalization and easier skill re-use we should decouple these into paradigms and try to align them later.

In passive pre-training, the agent learns about the world by just observing others. This stage of pre-training should essentially teach the agent about how the world dynamics work. There has not been much focus on passive pre-training over the years and with the recent advent of large datasets relevant for control like Ego4D[36], HM3D[101], and MT-Opt[56], I think the time is ripe to start leveraging real-world data for learning behavioral representation for control. One idea in this direction is to use masked trajectory modeling, a very successful pre-training objective in language modeling in BERT [18] and even more so recently in vision in ImageMAE[46], on observed trajectories to learn skills and sub-tasks that are permissible in the world. The passive pre-training itself has the potential to enable a lot of new applications: few-shot imitation learning and RL, action recognition, trajectory retrieval, behavioral prediction, etc.

In active pre-training, the agent learns about the world and its impact by acting in the environment. The ideas have been recently proliferated in the context of active pre-training [71, 67] which has shown promising signs of life for this paradigm of learning.

I think that decoupling the two learning paradigms and performing active pre-training on agents that have been already passively pre-trained is an interesting future work that can

enable truly general purpose control agents that can seamlessly adapt to the ever-changing environment dynamics in the real world with minimal supervision.

Bibliography

[1] Anurag Ajay et al. "{OPAL}: Offline Primitive Discovery for Accelerating Offline Reinforcement Learning". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=V69LGwJ0lIN.

[2] Vincentelli et al. "Support vector machines for analog circuit performance representation". In: (2003), pp. 964–969.

[3] G. Alpaydin, S. Balkir, and G. Dundar. "An evolutionary approach to automatic synthesis of high-performance analog integrated circuits". In: *IEEE Transactions on Evolutionary Computation* 7.3 (June 2003), pp. 240–252.

[4] Shikhar Bahl et al. "Neural dynamic policies for end-to-end sensorimotor learning". In: *arXiv preprint arXiv:2012.02788* (2020).

[5] Manuel F. M. Barros, Jorge M. C. Guilherme, and Nuno C. G. Horta. *Analog Circuits and Systems Optimization based on Evolutionary Computation Techniques*. Vol. 294. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2010.

[6] Aude Billard et al. *Survey: Robot programming by demonstration*. Tech. rep. Springrer, 2008.

[7] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[8] Pieter-Tjerk de Boer et al. "A Tutorial on the Cross-Entropy Method". In: *Annals of Operations Research* 134.1 (Feb. 2005), pp. 19–67.

[9] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[10] Tom B Brown et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[11] Roberto Calandra et al. "Manifold Gaussian processes for regression". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016, pp. 3338–3345.

[12] Eric Chang et al. "BAG2: A process-portable framework for generator-based AMS circuit design". In: *2018 IEEE Custom Integrated Circuits Conference (CICC)*. San Diego, CA: IEEE, Apr. 2018, pp. 1–8.

[13] Konstantinos Chatzilygeroudis et al. "A survey on policy search algorithms for learning robot controllers in a handful of trials". In: *IEEE Transactions on Robotics* 36.2 (2019), pp. 328–347.

[14] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *International conference on machine learning*. 2020.

[15] Andrea Coraddu et al. "Machine learning approaches for improving condition-based maintenance of naval propulsion plants". In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 230.1 (2016), pp. 136–153.

[16] Antoine Cully et al. "Robots that can adapt like animals". In: *Nature* 521.7553 (2015), pp. 503–507.

[17] Marc G R Degrauwe et al. "IDAC: An Interactive Design Tool for Analog CMOS Circuits". In: ().

[18] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[19] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[20] Jeffrey Donahue et al. "Long-term recurrent convolutional networks for visual recognition and description". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2625–2634.

[21] Yan Duan et al. "One-Shot Imitation Learning". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 1087–1098. URL: https://proceedings.neurips.cc/paper/2017/hash/ba3866600c3540f67c1e9575e213be0a-Abstract.html.

[22] Yan Duan et al. "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv:1611.02779* (2016).

[23] Scott Emmons et al. "Sparse Graphical Memory for Robust Planning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/385822e359afa26d52b5b286226f2cea-Abstract.html.

[24] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. "Scalable meta-learning for Bayesian optimization". In: *arXiv preprint arXiv:1802.02219* (2018).

[25] Matthias Feurer, Jost Springenberg, and Frank Hutter. "Initializing bayesian hyperparameter optimization via meta-learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[26] Matthias Fey and Jan Eric Lenssen. "Fast graph representation learning with PyTorch Geometric". In: *arXiv preprint arXiv:1903.02428* (2019).

[27] Chelsea Finn et al. "One-Shot Visual Imitation Learning via Meta-Learning". In: *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 357–368. URL: `http://proceedings.mlr.press/v78/finn17a.html`.

[28] Peter I Frazier. "A tutorial on Bayesian optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[29] Justin Fu et al. "D4rl: Datasets for deep data-driven reinforcement learning". In: *arXiv preprint arXiv:2004.07219* (2020).

[30] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: (June 2015). URL: `http://arxiv.org/abs/1506.02142`.

[31] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. "Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes". In: *Neurocomputing* 380 (2020), pp. 20–35.

[32] G. Gielen, P. Wambacq, and W.M. Sansen. "Symbolic analysis methods and applications for analog circuits: a tutorial overview". In: *Proceedings of the IEEE* 82.2 (Feb. 1994), pp. 287–304.

[33] G. Gielen et al. "An analogue module generator for mixed analogue/digital asic design". In: *International Journal of Circuit Theory and Applications* 23.4 (July 1995), pp. 269–283.

[34] Daniel Golovin et al. "Google vizier: A service for black-box optimization". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 1487–1495.

[35] Franz Graf et al. "2d image registration in ct images using radial image descriptors". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2011, pp. 607–614.

[36] Kristen Grauman et al. "Ego4d: Around the world in 3,000 hours of egocentric video". In: *arXiv preprint arXiv:2110.07058* (2021).

[37] Abhishek Gupta et al. "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning". In: *arXiv preprint arXiv:1910.11956* (2019).

[38] Kourosh Hakhamaneshi et al. "BagNet: Berkeley analog generator with layout optimizer boosted with deep neural networks". In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2019, pp. 1–8.

[39] Kourosh Hakhamaneshi et al. "Hierarchical Few-Shot Imitation with Skill Transition Models". In: *arXiv preprint arXiv:2107.08981* (2021).

[40] Kourosh Hakhamaneshi et al. "JUMBO: Scalable Multi-task Bayesian Optimization using Offline Data". In: *arXiv preprint arXiv:2106.00942* (2021).

[41] Kourosh Hakhamaneshi et al. "Pretraining Graph Neural Networks for few-shot Analog Circuit Modeling and Design". In: *arXiv preprint arXiv:2203.15913* (2022).

[42] Nikolaus Hansen. "The CMA evolution strategy: A tutorial". In: *arXiv preprint arXiv:1604.00772* (2016).

[43] R. Harjani, R.A. Rutenbar, and L.R. Carley. "OASYS: a framework for analog circuit synthesis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 8.12 (Dec. 1989), pp. 1247–1266.

[44] J.P. Harvey, M.I. Elmasry, and B. Leung. "STAIC: an interactive framework for synthesizing CMOS and BiCMOS analog circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11.11 (Nov. 1992), pp. 1402–1417.

[45] Mohsen Hassanpourghadi et al. "Circuit Connectivity Inspired Neural Network for Analog Mixed-Signal Functional Modeling". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2021, pp. 505–510.

[46] Kaiming He et al. "Masked autoencoders are scalable vision learners". In: *arXiv preprint arXiv:2111.06377* (2021).

[47] Kaiming He et al. "Momentum contrast for unsupervised visual representation learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738.

[48] Kaiming He et al. "Momentum contrast for unsupervised visual representation learning". In: *Conference on Computer Vision and Pattern Recognition*. 2020.

[49] M.delM. Hershenson, S.P. Boyd, and T.H. Lee. "Optimal design of a CMOS op-amp via geometric programming". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20 (Jan. 2001).

[50] Irina Higgins et al. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: (2016).

[51] Weihua Hu et al. "Open graph benchmark: Datasets for machine learning on graphs". In: *arXiv preprint arXiv:2005.00687* (2020).

[52] Wenbing Huang et al. "Scalable gaussian process regression using deep neural networks". In: *Twenty-fourth international joint conference on artificial intelligence*. Citeseer. 2015.

[53] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[54] Andrew Jaegle et al. "Perceiver io: A general architecture for structured inputs & outputs". In: *arXiv preprint arXiv:2107.14795* (2021).

[55] Paras Jain et al. "Representing Long-Range Context for Graph Neural Networks with Global Attention". In: *Advances in Neural Information Processing Systems* 34 (2021).

[56] Dmitry Kalashnikov et al. "Mt-opt: Continuous multi-task robotic reinforcement learning at scale". In: *arXiv preprint arXiv:2104.08212* (2021).

[57] Dmitry Kalashnikov et al. "Scalable deep reinforcement learning for vision-based robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2018, pp. 651–673.

[58] Kirthevasan Kandasamy et al. "Multi-fidelity gaussian process bandit optimisation". In: *Journal of Artificial Intelligence Research* 66 (2019), pp. 151–196.

[59] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[60] Aaron Klein and Frank Hutter. "Tabular benchmarks for joint architecture and hyperparameter optimization". In: *arXiv preprint arXiv:1905.04970* (2019).

[61] Aaron Klein et al. "Fast bayesian optimization of machine learning hyperparameters on large datasets". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 528–536.

[62] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-shot Image Recognition". In: 2015.

[63] H.Y. Koh, C.H. Sequin, and P.R. Gray. "OPASYN: a compiler for CMOS operational amplifiers". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 9.2 (Feb. 1990), pp. 113–125.

[64] Michael Krasnicki et al. "MAELSTROM: Efficient Simulation-Based Synthesis for Custom Analog Cells". In: ().

[65] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2012.

[66] Wim Kruiskamp and Domine Leenaerts. "DARWIN: CMOS opamp Synthesis by means of a Genetic Algorithm". In: ().

[67] Michael Laskin et al. "CIC: Contrastive Intrinsic Control for Unsupervised Skill Discovery". In: *arXiv preprint arXiv:2202.00161* (2022).

[68] Guohao Li et al. "Deepergcn: All you need to train deeper gcns". In: *arXiv preprint arXiv:2006.07739* (2020).

[69] Lisha Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.

[70] Yaping Li et al. "An artificial neural network assisted optimization system for analog design space exploration". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2019), pp. 2640–2653.

[71] Hao Liu and Pieter Abbeel. "Behavior from the void: Unsupervised active pre-training". In: *Advances in Neural Information Processing Systems* 34 (2021).

[72] Kara Liu et al. "Hallucinative Topological Memory for Zero-Shot Visual Planning". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 6259–6270. URL: `http://proceedings.mlr.press/v119/liu20h.html`.

[73] Mingjie Liu et al. "Parasitic-Aware Analog Circuit Sizing with Graph Neural Networks and Bayesian Optimization". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1372–1377.

[74] Yuzhe Ma et al. "High performance graph convolutional networks with applications in testability analysis". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–6.

[75] Viktor Makoviychuk et al. "Isaac gym: High performance gpu-based physics simulation for robot learning". In: *arXiv preprint arXiv:2108.10470* (2021).

[76] Azalia Mirhoseini et al. "A graph placement methodology for fast chip design". In: *Nature* 594.7862 (2021), pp. 207–212.

[77] Ashvin Nair et al. *AWAC: Accelerating Online Reinforcement Learning with Offline Datasets*. 2020. eprint: `arXiv:2006.09359`.

[78] Suraj Nair et al. "R3M: A Universal Visual Representation for Robot Manipulation". In: *arXiv preprint arXiv:2203.12601* (2022).

[79] Radford M. Neal. *Bayesian Learning for Neural Networks*. Ed. by P. Bickel et al. Vol. 118. Lecture Notes in Statistics. New York, NY: Springer New York.

[80] Andrew Y. Ng and Stuart J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. Ed. by Pat Langley. Morgan Kaufmann, 2000, pp. 663–670.

[81] W. Nye et al. "DELIGHT.SPICE: an optimization-based system for the design of integrated circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 7 (Apr. 1988), pp. 501–519.

[82] E.S. Ochotta, R.A. Rutenbar, and L.R. Carley. "Synthesis of high-performance analog circuits in ASTRX/OBLX". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15 (Mar. 1996), pp. 273–294.

[83] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748* (2018).

[84] Takayuki Osa et al. "An algorithmic perspective on imitation learning". In: *arXiv preprint arXiv:1811.06711* (2018).

[85] Simone Parisi et al. "The Unsurprising Effectiveness of Pre-Trained Vision Models for Control". In: *arXiv preprint arXiv:2203.03580* (2022).

[86] Rémi Pautrat, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. "Bayesian optimization with automatic prior selection for data-efficient direct policy search". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7571–7578.

[87] Xue Bin Peng et al. "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning". In: *CoRR* abs/1910.00177 (2019). arXiv: 1910.00177. URL: http://arxiv.org/abs/1910.00177.

[88] Valerio Perrone et al. "Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning". In: *arXiv preprint arXiv:1909.12552* (2019).

[89] Valerio Perrone et al. "Scalable hyperparameter transfer learning". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 6846–6856.

[90] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. "Accelerating Reinforcement Learning with Learned Skill Priors". In: *Conference on Robot Learning (CoRL)*. 2020.

[91] Johann Petrak. "Fast subsampling performance estimates for classification algorithm selection". In: *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*. 2000, pp. 3–14.

[92] Rodney Phelps and James R Hellums. "Anaconda: Simulation-Based Synthesis of Analog Circuits Via Stochastic Pattern Search". In: 19 ().

[93] Matthias Poloczek, Jialei Wang, and Peter I Frazier. "Warm starting Bayesian optimization". In: *2016 Winter Simulation Conference (WSC)*. IEEE. 2016, pp. 770–781.

[94] Dean Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*. Ed. by David S. Touretzky. Morgan Kaufmann, 1988, pp. 305–313. URL: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.

[95] Pankaj P. Prajapati and Mihir V. Shah. "Two stage CMOS operational amplifier design using particle swarm optimization algorithm". In: IEEE, Dec. 2015.

[96] Thomas Quarles et al. *SPICE 3 Version 3F5 User's Manual*. 1994.

[97] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).

[98] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[99] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. eprint: `arXiv:2103.00020`.

[100] Maithra Raghu et al. "Transfusion: Understanding transfer learning for medical imaging". In: *Advances in neural information processing systems* 32 (2019).

[101] Santhosh K Ramakrishnan et al. "Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI". In: *arXiv preprint arXiv:2109.08238* (2021).

[102] PS Rana. "Physicochemical properties of protein tertiary structure data set". In: *UCI Machine Learning Repository* (2013).

[103] Carl Edward Rasmussen. "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[104] Haoxing Ren et al. "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6.

[105] Stephane Ross, Geoffrey J. Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. Ed. by Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudik. Vol. 15. JMLR Proceedings. JMLR.org, 2011, pp. 627–635. URL: `http://proceedings.mlr.press/v15/ross11a/ross11a.pdf`.

[106] Rob A. Rutenbar, Georges G. E. Gielen, and Brian A. Antao. *Computer-Aided Design of Analog Integrated Circuits and Systems*. IEEE, 2002.

[107] David Salinas, Huibin Shen, and Valerio Perrone. "A quantile-based approach for hyperparameter transfer learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8438–8448.

[108] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. "Semi-parametric topological memory for navigation". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: `https://openreview.net/forum?id=SygwwGbRW`.

[109] Keertana Settaluri et al. "AutoCkt: deep reinforcement learning of analog circuit designs". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 490–495.

[110] Dhruv Shah et al. "ViNG: Learning Open-World Navigation with Visual Goals". In: *arXiv preprint arXiv:2012.09812* (2020).

[111] Bobak Shahriari et al. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[112] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.4 (2016), pp. 640–651.

[113] Alistair Shilton et al. "Regret bounds for transfer learning in Bayesian optimisation". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 307–315.

[114] Avi Singh et al. "Parrot: Data-Driven Behavioral Priors for Reinforcement Learning". In: *International Conference on Learning Representations*. 2020.

[115] Jasper Snoek et al. "Scalable bayesian optimization using deep neural networks". In: *International conference on machine learning*. PMLR. 2015, pp. 2171–2180.

[116] Jost Tobias Springenberg et al. "Bayesian optimization with robust Bayesian neural networks". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 4141–4149.

[117] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. "CURL: Contrastive Unsupervised Representations for Reinforcement Learning". In: *International Conference on Machine Learning*. 2020.

[118] Niranjan Srinivas et al. "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design". In: *Proceedings of the 27th International Conference on Machine Learning*. Omnipress. 2010.

[119] Freek Stulp and Olivier Sigaud. "Robot skill learning: From reinforcement learning to evolution strategies". In: *Paladyn, Journal of Behavioral Robotics* 4.1 (2013), pp. 49–61.

[120] Kevin Swersky, Jasper Snoek, and Ryan P Adams. "Multi-Task Bayesian Optimization". In: *Advances in Neural Information Processing Systems* 26 (2013), pp. 2004–2012.

[121] Yuval Tassa et al. "Deepmind control suite". In: *arXiv preprint arXiv:1801.00690* (2018).

[122] A. Torralba, J. Chavez, and L.G. Franquelo. "FASY: a fuzzy-logic based tool for analog synthesis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15 (July 1996), pp. 705–715.

[123] Athanasios Tsanas et al. "Enhanced classical dysphonia measures and sparse regression for telemonitoring of Parkinson's disease progression". In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2010, pp. 594–597.

[124] Petar Veličkovic et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[125] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[126] Oriol Vinyals et al. "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee et al. 2016, pp. 3630–3638. URL: https://proceedings.neurips.cc/paper/2016/hash/90e1357833654983612fb05e3ec9148c-Abstract.html.

[127] Michael Volpp et al. "Meta-learning acquisition functions for transfer learning in bayesian optimization". In: *arXiv preprint arXiv:1904.02642* (2019).

[128] P. Wambacq et al. "Efficient symbolic computation of approximated small-signal characteristics of analog integrated circuits". In: *IEEE Journal of Solid-State Circuits* 30.3 (Mar. 1995), pp. 327–330.

[129] Yaqing Wang et al. "Generalizing from a Few Examples: A Survey on Few-shot Learning". In: *ACM Comput. Surv.* 53.3 (2020), 63:1–63:34. DOI: 10.1145/3386252. URL: https://doi.org/10.1145/3386252.

[130] Chris Williams, Edwin V Bonilla, and Kian M Chai. "Multi-task Gaussian process prediction". In: *Advances in neural information processing systems* (2007), pp. 153–160.

[131] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Learning hyperparameter optimization initializations". In: *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE. 2015, pp. 1–10.

[132] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Scalable gaussian process-based transfer surrogates for hyperparameter optimization". In: *Machine Learning* 107.1 (2018), pp. 43–78.

[133] G.A Wolfe. "Performance macro-modeling techniques for fast analog circuit synthesis. Ph.D. dissertation, Dept. of Electrical and Computer Engineering and Computer Science, College of Engineering, University of Cincinnati, USA". PhD thesis. 1999.

[134] Glenn Wolfe and Ranga Vemuri. "Extraction and use of neural network models in automated synthesis of operational amplifiers". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22.2 (2003), pp. 198–212.

[135] Yifan Wu, George Tucker, and Ofir Nachum. "Behavior Regularized Offline Reinforcement Learning". In: *CoRR* abs/1911.11361 (2019). arXiv: 1911.11361. URL: http://arxiv.org/abs/1911.11361.

[136] Guo Zhang, Hao He, and Dina Katabi. "Circuit-GNN: Graph neural networks for distributed circuit design". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7364–7373.

[137] Tianhao Zhang et al. "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation". In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 1–8. DOI: 10.1109/ICRA.2018.8461249. URL: https://doi.org/10.1109/ICRA.2018.8461249.