UCLA
UCLA Electronic Theses and Dissertations

Title
Automated Machine Learning in the Era of Large Foundation Models

Permalink
https://escholarship.org/uc/item/1vc4421f

Author
Wang, Ruochen

Publication Date
2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California

Los Angeles

Automated Machine Learning in the Era of Large Foundation Models

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Ruochen Wang

2024

© Copyright by

Ruochen Wang

2024

Abstract of the DISSERTATION

Automated Machine Learning in the Era of Large Foundation Models

by

Ruochen Wang

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2024

Professor Cho-Jui Hsieh, Co-Chair

Professor Wei Wang, Co-Chair

Intelligence, one of the most profound phenomena on Earth, has evolved over 600 million years, transforming from simple neural systems into human cognition capable of unraveling universal mysteries and creating silicon-based intelligence. This evolutionary process, with its inherent drive towards increasing complexity, seemingly defies thermodynamic principles, suggesting the existence of self-evolving mechanisms in life. Recreating such mechanisms within artificial systems is a crucial milestone on the path toward Artificial General Intelligence (AGI).

Automated Machine Learning (AutoML) represents a significant step in this direction. By enabling AI systems to optimize their own design processes, AutoML reformulates machine learning pipeline construction as a search problem, automating the selection of architectures, optimizers, hyperparameters, and even reasoning paths from an expansive search space. Despite its current limitations, AutoML has already demonstrated remarkable success in advancing machine learning across diverse applications.

This thesis delves into the synergistic interplay between AutoML and large foundation models, particularly the recent breakthroughs in large-scale generative models like large language models (LLMs) and diffusion models. These models exhibit emergent behaviors, highlighting machine learning systems as complex entities where scaling produces unpredictable and potentially transformative capabilities. We emphasize the application of AutoML in automating the design of training and inference processes, crucial for the continued advancement of these models.

Ultimately, we aspire that this research contributes a meaningful step towards the ambitious pursuit of Artificial General Intelligence (AGI).

The dissertation of Ruochen Wang is approved.

Aditya Grover

Baharan Mirzasoleiman

Wei Wang, Committee Co-Chair

Cho-Jui Hsieh, Committee Co-Chair

University of California, Los Angeles

2024

To my parents

TABLE OF CONTENTS

List of Figures

List of Tables

Acknowledgments

I am deeply grateful to my advisor, Cho-Jui Hsieh, for his invaluable guidance and unwavering support throughout my research. I also extend my sincere thanks to all those who have contributed to my academic journey, including collaborators, mentors, colleagues, and committee members. Their insights and encouragement have been essential to my development. Finally, I wish to express my profound gratitude to my family, especially my parents, for their unwavering love and support during challenging times.

Thank you,
どうもありがとうございます,
대단히감사합니다,
非常感谢!

Vita

2013–2015	B.S. (Finance Honors Class), Shanghai University of Finance and Economics.
2015–2019	B.S. (Computer Science, Statistics), The University of Michigan, Ann Arbor.
2020–present	Research Assistant, Computer Science Department, UCLA.

CHAPTER 1

Introduction

Intelligence, and how it arises, is arguably one of the most fascinating phenomena taking place on planet Earth. Within 600 million years of real physical simulation, intelligence has evolved from the most basic neuro-system that only handles basic coordination of movements, to the vastly more intricate human brains that can unravel the mystery of the universe, build spaceships for interplanetary exploration, and even create silicon-based intelligence that surpasses ourselves. Such a process of increasing complexity seems rather unorthodox, as the second law of thermodynamics suggests the inevitable increase of entropy in a closed system [Sch44]. This "paradox" implies the existence of a self-evolving mechanism in life forms, which, once the start button is hit, will not stop sucking negative entropy from the environment to maintain and increase its order. Re-creating this self-evolving mechanism is a potential milestone on the path to Artificial General Intelligence (AGI), assuming that AGI is ever possible.

Automated machine learning (AutoML) provides one of the first practical steps toward building such a self-evolving mechanism. At the core, AutoML aims to leverage the power of AI to develop itself. It formulates the design of machine learning pipeline into a search problem, where the goal is to identify the best-performing architectures [CWC21, WCC20, HWL21, WCC21], optimizers [BZV17], hyperparameters [XLC22] and inference processes [WAC24, WLH24, WSY24] for various models and tasks, from a vast pool of candidates. Although existing formulation of AutoML is far from "self-evolving", it still witnessed numerous real-world applications in advancing the design of machine learning pipelines, until the recent breakthrough in large language models.

Recent advancements in large-scale text-based generative models have transformed how we approach many problems in deep learning. Noticeably, large language models (LLM) prove that the current paradigm of machine learning is a specific complex system, where, by merely increasing the number of its basic units (model and dataset size), unpredictable features emerge [Mit09].

This thesis explores the interplay between AutoML and large foundation models, focusing on how search-based methods can automate the design of architectures, optimizers and inference processes. We re-examine the role of AutoML in this new era, demonstrating its resurgence in the form of prompt optimization. We argue that prompt optimization is a form of "model selection" where different prompts effectively select different "submodels" within a pretrained LLM.

The rest of the paper is organized as two part. In Part I, we discuss methodology for developing efficient search algorithms (Chapter 2) and generic search spaces (Chapter 3) that enable automated design of novel architectures and optimizers. In Part II, we discuss how search-based techniques can unleash the potential of large foundation models through prompt optimization. We discuss prompt optimization methods for LLM in Chapter 5, and diffusion models in Chapter 6.

Part I

Training: Search for Architectures and Optimizers

CHAPTER 2

Efficient and Robust Search Algorithm for Architectures

2.1 Problem Statement

Neural Architecture Search (NAS) has been drawing increasing attention in both academia and industry for its potential to automatize the process of discovering high-performance architectures, which have long been handcrafted. Early works on NAS deploy Evolutionary Algorithm [SM02, RMS17, LSV17] and Reinforcement Learning [ZL17, PGZ18, ZYW18] to guide the architecture discovery process. These pioneering methods require training a large number of discrete architectures, and hence incur significant amount of computation costs. In the attempt to improve the efficiency of NAS, several one-shot weight-sharing methods have been proposed that dramatically cut down the search cost [BLR18, GZM19, BKZ18].

As a particularly popular instance of one-shot methods, DARTS [LSY19b] enables the search process to be performed with a gradient-based optimizer in an end-to-end manner. It applies continuous relaxation that transforms the categorical choice of architectures into continuous architecture parameters α. The resulting supernet can be optimized via gradient-based methods, and the operations associated with the largest architecture parameters are selected to form the final architecture. Despite its simplicity, several works cast doubt on the effectiveness of DARTS. For example, a simple randomized search [LT19] outperforms the original DARTS; [ZES20] observes that DARTS degenerates to networks filled with parametric-free operations such as the skip connection or even random noise, leading to the poor performance of the selected architecture.

While the majority of previous research attributes the failure of DARTS to its supernet optimization [ZES20, CH20a, CWC21], little has been discussed about the validity of another important assumption: *the value of α reflects the strength of the underlying operations.* In this paper, we conduct an in-depth analysis of this problem. Surprisingly, we find that in many cases, α does not really indicate the operation importance in a supernet. Firstly, the operation associated with larger α does not necessarily result in higher validation accuracy after discretization. Secondly, as an important example, we show mathematically that the domination of skip connection observed in DARTS (i.e. α_{skip} becomes larger than other operations.) is in fact a reasonable outcome of the supernet's optimization but becomes problematic when we rely on α to select the best operation.

If α is not a good indicator of operation strength, how should we select the final architecture from a pretrained supernet? We discuss two lines of solutions. The first line bypasses α and directly measures the operation strength based on its contribution to the supernet performance in an adversarial fashion. Concretely, we propose an alternative perturbation-based architecture selection method. Given a pretrained supernet, the best operation on an edge is selected and discretized based on how much it perturbs the supernet accuracy; The final architecture is derived edge by edge, with fine-tuning in between so that the supernet remains converged for every operation decision. The second line aims at aligning the architecture selection with differentiable architecture search, by formulating architecture search as a distribution learning problem, which induces implicit hessian regularization [CWC21]. Empirical evaluations show that both lines of methods are able to drastically improve the search performance and the robustness of differentiable NAS.

To make the picture a bit more complete, the remaining chapters of the paper discusses an alternative architecture search paradigm that does not rely on weight-sharing supernet to make search decisions, called predictor-based NAS. Predictor-based methods use a tiny subset of architectures to train a surrogate model to predict the accuracy of each individual architecture. This line of methods do not rely on a weight-sharing one-shot supernet, and thus is free from the various inductive biases of differentiable NAS methods.

2.2 Understanding Differentiable NAS

Despite the search efficiency of Differentiable Neural Architecture Search (DARTS), several work finds that it generalizes poorly to a wide range of search spaces (Section 2.4). For example, [ZES20] observes that DARTS degenerates to networks filled with parametric-free operations such as the skip connection or even random noise, leading to the poor performance of the selected architecture. While previous analysis attributes the poor generalization of DARTS to the failure of supernet optimization, we show both empirically and mathematically that it is in fact caused by the magnitude-based architecture selection method (Section 2.5).

2.3 Differentiable Architecture Search Framework

We start by reviewing the formulation of DARTS. DARTS' search space consists of repetitions of cell-based microstructures. Every cell can be viewed as a DAG with N nodes and E edges, where each node represents a latent feature map x^i, and each edge is associated with an operation o (e.g. *skip_connect*, *sep_conv_3x3*) from the search space $\mathcal{O}$. Continuous relaxation is then applied to this search space: Concretely, every operation on an edge is activated during the search phase, with their outputs mixed by the architecture parameter α to form the final mixed output of that edge $\bar{m}(x^i) = \sum_{o \in \mathcal{O}} \frac{\exp \alpha_o}{\sum_{o'} \exp \alpha_{o'}} o(x^i)$. This particular formulation allows the architecture search to be performed in a differentiable manner: DARTS jointly optimizes α and model weight w with the following bilevel objective via alternative gradient updates:

$$\min_{\alpha} \ \mathcal{L}_{val}(w^*, \alpha) \qquad \text{s.t.} \qquad w^* = \arg\min_{w} \ \mathcal{L}_{train}(w, \alpha). \tag{2.1}$$

We refer to the continuous relaxed network used in the search phase as the **supernet** of DARTS. At the end of the search phase, the operation associated with the largest α_o on each edge will be selected from the supernet to form the final architecture. An important implicit assumption here is that the magnitude of $\alpha = (\alpha_o)_o$ represents the strength of the underlying operation.

2.4 Failure Mode Analysis of DARTS

Several works cast doubt on the generalization of DARTS. [ZES20] tests DARTS on four different search spaces and observes significantly degenerated performance, resulting in architectures filled with skip connections. They empirically find that the selected architectures perform poorly when DARTS' supernet falls into high curvature areas of validation loss (captured by large dominant eigenvalues of the Hessian $\nabla^2_{\alpha,\alpha}\mathcal{L}_{val}(w, \alpha)$). While [ZES20] relates this problem to the failure of supernet training in DARTS, we examine it from the architecture selection aspects of DARTS, and show that much of DARTS' robustness issue can be alleviated by a better architecture selection method.

2.5 The pitfall of magnitude-based architecture selection in DARTS

In this section, we put forward the opinion that the architecture parameter α does not necessarily represent the strength of the underlying operation in general, in direct contradiction to the claim made in DARTS. As an important example, we mathematically justify that the skip connection domination phenomenon observed in DARTS is reasonable by itself, and becomes problematic when combined with the magnitude-based architecture selection.

2.5.1 α may not represent the operation strength

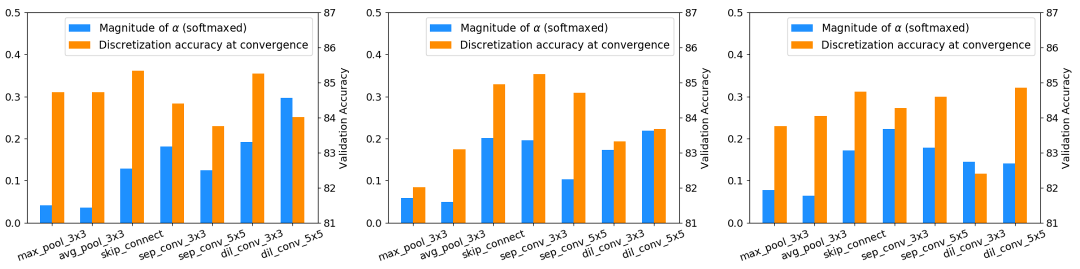
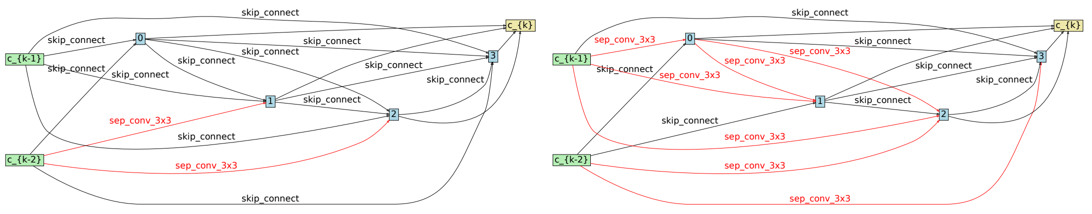


Figure 2.1: α vs discretization accuracy at convergence of all operations on 3 randomly selected edges from a pretrained DARTS supernet (one subplot per edge). The magnitude of α for each operation does not necessarily agree with its relative discretization accuracy at convergence.

Following DARTS, existing differentiable NAS methods use the value of architecture parameters α to select the final architecture from the supernet, with the implicit assumption that α represents the strength of the underlying operations. In this section, we study the validity of this assumption in detail.



(a) Magnitude (b) Strength

Figure 2.2: Operation strength on each edge of S2 *(skip_connect, sep_conv_3x3)*. (a). Operations associated with the largest α. (b). Operations that result in the highest discretization validation accuracy at convergence. Parameterized operations are marked red.

Consider one edge on a pretrained supernet; the strength of an operation on the edge can be naturally defined as the supernet accuracy after we discretize to this operation and fine-tune the remaining network until it converges again; we refer to this as "discretization accuracy at convergence" for short. The operation that achieves the best discretization accuracy at convergence can be considered as the best operation for the given edge. Figure 2.1 shows the comparison of α (blue) and operation strength (orange) of randomly select edges on DARTS supernet. As we can see, the magnitude of α for each operation does not necessarily agree with their relative strength measured by discretization accuracy at convergence. Moreover, operations assigned with small αs are sometimes strong ones that lead to high discretization accuracy at convergence. To further verify the mismatch, we investigate the operation strength on search space S2, where DARTS fails dramatically due to excessive skip connections [ZES20]. S2 is a variant of DARTS search space that only contains two operations per edge (*skip_connect, sep_conv_3x3*). Figure 2.2 shows the selected operations based on α (left) and operation strength (right) on all edges on S2. From Figure 2.2a, we can see that $\alpha_{skip\_connect} > \alpha_{sep\_conv\_3x3}$ on 12 of 14 edges. Consequently, the derived child architecture will lack representation ability and perform poorly due to too many skip connections. However, as shown in Figure 2.2b, the supernet benefits more from discretizing to *sep_conv_3x3* than *skip_connect* on half of the edges.

There are several reason why α fails to capture the operation strength. Firstly, consider the second order approximation of the validation loss of a pretrained supernet:

$$\mathcal{L}_{val}(\hat{\alpha}, w) \approx \mathcal{L}_{val}(\alpha, w) + (\hat{\alpha} - \alpha)^T \nabla_\alpha \mathcal{L}_{val}(\alpha, w) + \frac{1}{2}(\hat{\alpha} - \alpha)^T \nabla^2_{\alpha^2} \mathcal{L}_{val}(\alpha, w)(\hat{\alpha} - \alpha) \tag{2.2}$$

$$= \mathcal{L}_{val}(\alpha, w) + \frac{1}{2}(\hat{\alpha} - \alpha)^T \nabla^2_{\alpha^2} \mathcal{L}_{val}(\alpha, w)(\hat{\alpha} - \alpha) \tag{2.3}$$

$$(\nabla_\alpha \mathcal{L}_{val}(\alpha, w) = 0 \text{ at convergence})$$

Where $\hat{\alpha}$ is the perturbed architecture parameters and α is the current instance. We can see that the influence of $\hat{\alpha}$ on the supernet's validation loss depends not only on $\hat{\alpha}$ itself but also the curvature $\nabla^2_{\alpha^2} \mathcal{L}_{val}(\alpha, w)$. However, in magnitude-based architecture selection, the Hessian term is completely ignored, which corresponds to the case when $\nabla^2_{\alpha^2} \mathcal{L}_{val}(\alpha, w) = I$. Secondly, architecture parameters are interdependent: selecting and discretizing one edge to an operation modifies the supernet, thereby affecting subsequent selection.

2.5.2 A case study: skip connection

Several works point out that DARTS tends to assign large α to skip connections, resulting in shallow architectures with poor generability [ZES20, LZS19, BHX19]. This "skip connection domination" issue is generally attributed to the failure of DARTS' supernet optimization. In contrast, we draw inspiration from research on ResNet [HZR16] and show that this phenomenon by itself is a reasonable outcome while DARTS refines its estimation of the optimal feature map, rendering α_{skip} ineffective in the architecture selection.

Table 2.1: Test accuracy before and after layer (edge) shuffling on CIFAR-10. For ResNet and VGG, we randomly swap two layers in each stage (defined as successive layers between two downsampling blocks. For DARTS supernet, we randomly swap two edges in every cell.

	VGG	**ResNet**	**DARTS**
Before	92.69	93.86	88.44
After	9.83 ± 0.33	83.2015 ± 2.03	81.09 ± 1.87

In vanilla networks (e.g., VGG), each layer computes a new level of feature map from the output feature map of the predecessor layer; thus, reordering layers at test time would dramatically hurt the performance [VWB16]. Unlike vanilla networks, [GSS17] and [VWB16] discover that successive layers in ResNet with compatible channel sizes are in fact estimating the same optimal feature map so that the outputs of these layers stay relatively close to each other at convergence; As a result, ResNet's test accuracy remains robust under layer reordering. [GSS17] refers to this unique way of feature map estimation in ResNet as the "unrolled estimation."

DARTS' supernet resembles ResNet, rather than vanilla networks like VGG, in both appearance and behavior. Appearance-wise, within a cell of DARTS' supernet, edges with skip connection are in direct correspondence with the successive residual layers in ResNet. Behavior-wise, DARTS' supernet also exhibits a high degree of robustness under edge shuffling. As shown in Table 2.1, randomly reordering edges on a pretrained DARTS' supernet at test time also has little effect on its performance. This evidence indicates that DARTS performs unrolled estimation like ResNet as well, i.e., edges within a cell share the same optimal feature map that they try to estimate. In the following proposition, we apply this finding and provide the optimal solution of α in the sense of minimizing the variance of feature map estimation.

Proposition 1. [1] *Without loss of generality, consider one cell from a simplified*

[1]Proposition 1 unfolds the optimal α in principle and does not constraint the particular

search space consists of two operations: (skip, conv). Let m^* denotes the optimal feature map, which is shared across all edges according to the unrolled estimation view [GSS17]. Let $o_e(x_e)$ be the output of convolution operation, and let x_e be the skip connection (i.e., the input feature map of edge e). Assume m^*, $o_e(x_e)$ and x_e are normalized to the same scale. The current estimation of m^* can then be written as:

$$\overline{m}_e(x_e) = \frac{\exp(\alpha_{conv})}{\exp(\alpha_{conv}) + \exp(\alpha_{skip})} o_e(x_e) + \frac{\exp(\alpha_{skip})}{\exp(\alpha_{conv}) + \exp(\alpha_{skip})} x_e, \tag{2.4}$$

where α_{conv} and α_{skip} are the architecture parameters defined in DARTS. The optimal α^*_{conv} and α^*_{skip} minimizing $var(\overline{m}_e(x_e) - m^*)$, the variance of the difference between the optimal feature map m^* and its current estimation $\overline{m}_e(x_e)$, are given by:

$$\alpha^*_{conv} \propto var(x_e - m^*) \tag{2.5}$$

$$\alpha^*_{skip} \propto var(o_e(x_e) - m^*). \tag{2.6}$$

We refer the reader to the original paper of DARTS-PT [WCC20] for a detailed proof. From eq. (2.5) and eq. (2.6), we can see that the relative magnitudes of α_{skip} and α_{conv} come down to which one of x_e or $o_e(x_e)$ is closer to m^* in variance:

- x_e (input of edge e) comes from the mixed output of the previous edge. Since the goal of every edge is to estimate m^* (unrolled estimation), x_e is also directly estimating m^*.
- $o_e(x_e)$ is the output of a single convolution operation instead of the complete

optimization method (i.e., bilevel, single-level, or blockwise update) to achieve it. Moreover, this proposition can be readily extended to various other search spaces since we can group all non-skip operations into a single $o_e(\cdot)$.

mixed output of edge e, so it will deviate from m^* even at convergence.

Therefore, in a well-optimized supernet, x_e will naturally be closer to m^* than $o_e(x_e)$, causing α_{skip} to be greater than α_{conv}.

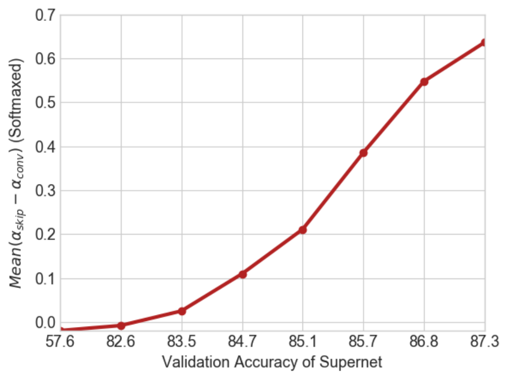


Figure 2.3: $mean(\alpha_{skip} - \alpha_{conv})$ (softmaxed) v.s. supernet's validation accuracy. The gap of $(\alpha_{skip} - \alpha_{conv})$ increases as supernet gets better.

Our analysis above indicates that the better the supernet, the larger the $(\alpha_{skip} - \alpha_{conv})$ gap (softmaxed) will become since x_e gets closer and closer to m^* as the supernet is optimized. This result is evidenced in Figure 2.3, where $mean(\alpha_{skip} - \alpha_{conv})$ continues to grow as the supernet gets better. In this case, although $\alpha_{skip} > \alpha_{conv}$ is reasonable by itself, it becomes an inductive bias to NAS if we were to select the final architecture based on α.

2.6 Improving the Effectiveness and Robustness of Differentiable NAS

In the previous chapter, we identify that the poor generalization of DARTS, including skip connection domination, is caused by the failure of magnitude-based architecture selection. Inspired by this analysis, we introduce two lines of methods to solve this problem: 1) A perturbation-based architecture selection that bypasses architecture parameter α and measures the operation strength in an adversarial fashion (Section 2.7), and 2) improve the alignment between α and operation strength via Hessian Regularization.

2.7 Perturbation-based architecture selection

Instead of relying on the α value to select the best operation, we propose to directly evaluate operation strength in terms of its contribution to the supernet's performance. The operation selection criterion is laid out in section 2.7.1. In section 2.7.2, we describe the entire architecture selection process.

2.7.1 Evaluating the strength of each operation

In section 2.5.1, we define the strength of each operation on a given edge as how much it contributes to the performance of the supernet, measured by discretization accuracy. To avoid inaccurate evaluation due to large disturbance of the supernet during discretization, we fine-tune the remaining supernet until it converges again, and then compute its validation accuracy (discretization accuracy at convergence). The fine-tuning process needs to be carried out for evaluating each operation on an edge, leading to substantial computation costs.

To alleviate the computational overhead, we consider a more practical measure of operation strength: for each operation on a given edge, we mask it out while keeping all other operations, and re-evaluate the supernet. The one that results in the largest drop in the supernet's validation accuracy will be considered as the most important operation on that edge. This alternative criterion incurs much less perturbation to the supernet than discretization since it only deletes one operation from the supernet at a time. As a result, the supernet's validation accuracy after deletion stays close to the unmodified supernet, and thus it alleviates the requirement of tuning the remaining supernet to convergence. Therefore, we implement this measurement for the operation selection in this work.

2.7.2 The complete architecture selection process

Our method operates directly on top of DARTS' pretrained supernet. Given a supernet, we randomly iterate over all of its edges. We evaluate each operation on an edge, and select the best one to be discretized based on the measurement described in section 2.7.1. After that, we tune the supernet for a few epochs to recover the accuracy lost during discretization. The above steps are repeated until all edges are decided. The cell topology is decided in a similar fashion. This simple method is termed "perturbation-based architecture selection (PT)" in the following sections.

2.7.3 Experimental Evaluation

In this section, we demonstrate that the perturbation-based architecture selection method is able to consistently find better architectures than those selected based on the values of α. The evaluation is based on the search space of DARTS and NAS-Bench-201 [DY20], and we show that the perturbation-based architecture selection method can be applied to several variants of DARTS.

2.7.3.1 Results on DARTS' CNN search space

We keep all the search and retrain settings identical to DARTS since our method only modifies the architecture selection part. After the search phase, we perform perturbation-based architecture selection on the pretrained supernet. We tune the supernet for 5 epochs between two selections as it is enough for the supernet to recover from the drop of accuracy after discretization. We run the search and architecture selection phase with four random seeds and report both the best and average test errors of the obtained architectures.

As shown in Table 2.2, the proposed method (DARTS+PT) improves DARTS' test error from 3.00% to 2.61%, with manageable search cost (0.8 GPU days). Note that by only changing the architecture selection method, DARTS performs significantly better than many other differentiable NAS methods that enjoy carefully designed optimization process of the supernet, such as GDAS [DY19] and SNAS [XZL18]. This empirical result suggests that architecture selection is crucial to DARTS: with the proper selection algorithm, DARTS remains a very competitive method.

Our method is also able to improve the performance of other variants of DARTS. To show this, we evaluate our method on SDARTS(rs) and SGAS [CH20a, LQD20]. SDARTS(rs) is a variant of DARTS that regularizes the search phase by applying Gaussian perturbation to α. Unlike DARTS and SDARTS, SGAS performs progressive search space shrinking. Concretely, SGAS progressively discretizes its edges with the order from most to least important, based on a novel edge importance score. For a fair comparison, we keep its unique search space shrinking process unmodified and only replace its magnitude-based operation selection with ours. As we can see from Table 2.2, our method consistently achieves better average test errors than its magnitude-based counterpart. Concretely, the proposed method improves SDARTS' test error from 2.67% to 2.54% and SGAS' test error from 2.66% to 2.56%. Moreover, the best architecture discovered in our experiments achieves a test error of 2.44%, ranked top among other NAS methods.

Table 2.2: Test error of architectures discovered by perturbation-based selection on DARTS Space and CIFAR-10.

Architecture	**Test Error (%)**	**Params (M)**	**Search Cost (GPU days)**	**Search Method**
DenseNet-BC [HLM17]	3.46	25.6	-	manual
NASNet-A [ZVS18]	2.65	3.3	2000	RL
AmoebaNet-A [RAH19]	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-B [RAH19]	2.55 ± 0.05	2.8	3150	evolution
PNAS [LZN18]*	3.41 ± 0.09	3.2	225	SMBO
ENAS [PGZ18]	2.89	4.6	0.5	RL
NAONet [LTQ18]	3.53	3.1	0.4	NAO
SNAS (moderate) [XZL18]	2.85 ± 0.02	2.8	1.5	gradient
GDAS [DY19]	2.93	3.4	0.3	gradient
BayesNAS [ZYW19]	2.81 ± 0.04	3.4	0.2	gradient
ProxylessNAS [CZH19]†	2.08	5.7	4.0	gradient
NASP [YXT20]	2.83 ± 0.09	3.3	0.1	gradient
P-DARTS [CXW19]	2.50	3.4	0.3	gradient
PC-DARTS [XXZ20]	2.57 ± 0.07	3.6	0.1	gradient
R-DARTS (L2) [ZES20]	2.95 ± 0.21	-	1.6	gradient
DARTS [LSY19b]	3.00 ± 0.14	3.3	0.4	gradient
SDARTS-RS [CH20a]	2.67 ± 0.03	3.4	0.4	gradient
SGAS (Cri 1. avg) [LQD20]	2.66 ± 0.24	3.7	0.25	gradient
DARTS+PT (avg)*	2.61 ± 0.08	3.0	$0.8^{\ddagger}$	gradient
DARTS+PT (best)	2.48	3.3	$0.8^{\ddagger}$	gradient
SDARTS-RS+PT (avg)*	2.54 ± 0.10	3.3	$0.8^{\ddagger}$	gradient
SDARTS-RS+PT (best)	2.44	3.2	$0.8^{\ddagger}$	gradient
SGAS+PT (Crit.1 avg)*	2.56 ± 0.10	3.9	$0.29^{\ddagger}$	gradient
SGAS+PT (Crit.1 best)	2.46	3.9	$0.29^{\ddagger}$	gradient

† Obtained on a different space with PyramidNet [HKK17] as the backbone.
‡ Recorded on a single GTX 1080Ti GPU.
* Obtained by running the search and retrain phase under four different seeds and computing the average test error of the derived architectures.

2.7.3.2 Performance on NAS-Bench-201 search space

To further verify the effectiveness of the proposed perturbation-based architecture selection, we conduct experiments on NAS-Bench-201. NAS-Bench-201 provides a unified cell-based search space similar to DARTS. Every architecture in the search space is trained under the same protocol on three datasets (cifar10, cifar100, and imagenet16-120), and their performance can be obtained by querying the database. As in section 2.7.3.1, we take the pretrained supernet from DARTS and apply our method on top of it. All other settings are kept unmodified. Figure 2.4 shows the performance trajectory of DARTS+PT compared with DARTS. While the architectures found by magnitude-based selection degenerates over time, the perturbation-based method is able to extract better architectures

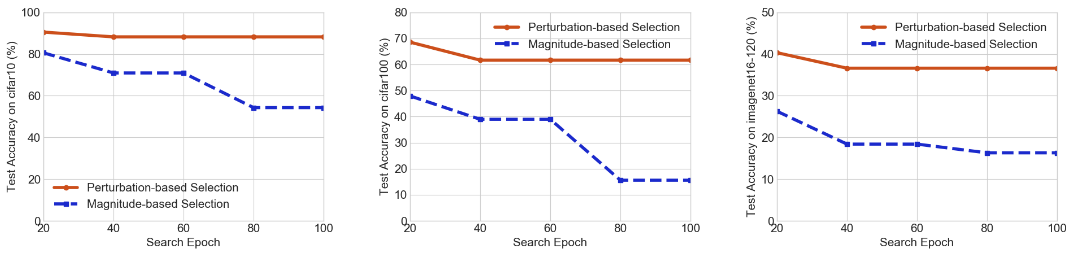


Figure 2.4: Trajectory of test accuracy on space NAS-Bench-201 and three datasets (Left: cifar10, Middle: cifar100, Right: Imagenet16-120). The test accuracy of our method is plotted by taking the snapshots of DARTS' supernet at corresponding epochs and run our selection method on top of it.

from the same underlying supernets stably. The result implies that the DARTS' degenerated performance comes from the failure of magnitude based architecture selection.

2.8 Aligning architecture parameter with operation strength via distribution learning

Recall that in Section 2.5, we show that the misalignment between α and operation strength is caused by the existence of the Hessian term $\nabla^2_{\alpha^2}\mathcal{L}_{val}(\alpha, w)$ (eq. 2.2). When this hessian is ill-conditioned, the magnitude of *alpha* deviates from the operation strength. Therefore, another way to improve the robustness of DARTS is to regularize this Hessian. In this section, we propose a principled method by formulating differentiable NAS as a distribution learning problem.

2.8.0.1 Neural Architecture Search as distribution learning

Bilevel-Optimization with Simplex Constraints In DARTS, the uncontrained architecture parameters are mapped to the operation mixing weight via softmax function, allowing it to lie in the probability simplex. To motivate our method, we first generalize the bilevel formulation of DARTS by using θ to simplex-constrained represent the operation mixing weight:

$$\min_{\theta} \ \mathcal{L}_{val}(w^*, \theta) \quad \text{s.t.} \ \ w^* = \arg\min_{w} \ \mathcal{L}_{train}(w, \theta), \quad \sum_{o=1}^{|\mathcal{O}|} \theta_o^{(i,j)} = 1, \ \forall \ (i,j), \ i < j, \tag{2.7}$$

where the simplex constraint $\sum_{o=1}^{|\mathcal{O}|} \theta_o^{(i,j)} = 1$ can be either solved explicitly via Lagrangian function [LKB20], or eliminated by substitution method (e.g., $\theta = Softmax(\alpha), \alpha \in \mathcal{R}^{|\mathcal{O}| \times |E|}$) [LSY19b].

Learning a Distribution over Operation Mixing Weight Previous differentiable architecture search methods view the operation mixing weight θ as learnable parameters that can be directly optimized [LSY19b, XXZ20, LKB20]. This has been shown to cause θ to overfit the validation set and thus induce large generalization error [BKZ18, ZSH20, CH20a]. We recognize that this treatment is equivalent to performing point estimation (e.g., MLE/MAP) of θ in probabilistic view, which is inherently prone to overfitting [Bis16, GCS04]. Based on this insight, we formulate the differentiable architecture search as a distribution learning problem. The operation mixing weight θ is treated as random variables sampled from a learnable distribution. Formally, let $q(\theta|\beta)$ denote the distribution of θ parameterized by β. The bi-level objective is then given by:

$$\min_{\beta} E_{q(\theta|\beta)}\big[\mathcal{L}_{val}(w^*, \theta)\big] + \lambda d(\beta, \hat{\beta}) \ \text{ s.t. } \ w^* = \arg\min_{w} \ \mathcal{L}_{train}(w, \theta). \tag{2.8}$$

where $d(\cdot, \cdot)$ is a distance function. Since θ lies on the probability simplex, we select Dirichlet distribution to model its behavior, i.e., $q(\theta|\beta) \sim Dir(\beta)$, where β represents the Dirichlet concentration parameter. Dirichlet distribution is a widely used distribution over the probability simplex [JLP19, Dav03, LHZ20, KNZ19], and it enjoys nice properties that enables gradient-based training [Mar18].

2.8.1 The implicit Regularization on Hessian

The objective in (2.8) can be viewed as a Lagrangian function of the following constraint objective:

$$\min_{\beta} E_{q(\theta|\beta)}\big[\mathcal{L}_{val}(w^*, \theta)\big] \quad \text{s.t.} \quad w^* = \arg\min_{w} \ \mathcal{L}_{train}(w, \theta) \ , \ d(\beta, \hat{\beta}) \le \delta, \tag{2.9}$$

Here we derive an approximated lower bound based on (2.9), which demonstrates that our method implicitly controls the conditional number of the Hessian matrix $\nabla^2_\theta \tilde{\mathcal{L}}_{val}(w, \theta)$.

Proposition 2. *Let $d(\beta, \hat{\beta}) = \|\beta - \hat{\beta}\|_2 \le \delta$ and $\hat{\beta} = 1$ in the bi-level formulation (2.9). Let μ denote the mean under the Laplacian approximation of Dirichlet. If $\nabla^2_\mu \tilde{\mathcal{L}}_{val}(w^*, \mu)$ is Positive Semi-definite, the upper-level objective can be approximated bounded by:*

$$E_{q(\theta|\beta)}(\mathcal{L}_{val}(w, \theta)) \gtrsim \tilde{\mathcal{L}}_{val}(w^*, \mu) + \frac{1}{2}\Big(\frac{1}{1+\delta}\big(1 - \frac{2}{|\mathcal{O}|}\big) + \frac{1}{|\mathcal{O}|}\frac{1}{1+\delta}\Big) tr\big(\nabla^2_\mu \tilde{\mathcal{L}}_{val}(w^*, \mu)\big) \tag{2.10}$$

with:

$$\tilde{\mathcal{L}}_{val}(w^*, \mu) = \mathcal{L}_{val}(w^*, Softmax(\mu)), \quad \mu_o = \log \beta_o - \frac{1}{|\mathcal{O}|}\sum_{o'} \log \beta_{o'}, \quad o = 1, \dots, |\mathcal{O}|.$$

This proposition is driven by the Laplacian approximation to the Dirichlet distribution [Mac98, Aka17]. The lower bound (2.10) indicates that minimizing

Table 2.3: Test error of the architectures discovered by DrNAS on DARTS Space and CIFAR-10.

Architecture	**Test Error (%)**	**Params (M)**	**Search Cost (GPU days)**	**Search Method**
DenseNet-BC [HLM17]*	3.46	25.6	-	manual
NASNet-A [ZVS18]	2.65	3.3	2000	RL
AmoebaNet-A [RAH19]	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-B [RAH19]	2.55 ± 0.05	2.8	3150	evolution
PNAS [LZN18]*	3.41 ± 0.09	3.2	225	SMBO
ENAS [PGZ18]	2.89	4.6	0.5	RL
DARTS (1st) [LSY19b]	3.00 ± 0.14	3.3	0.4	gradient
DARTS (2nd) [LSY19b]	2.76 ± 0.09	3.3	1.0	gradient
SNAS (moderate) [XZL18]	2.85 ± 0.02	2.8	1.5	gradient
GDAS [DY19]	2.93	3.4	0.3	gradient
BayesNAS [ZYW19]	2.81 ± 0.04	3.4	0.2	gradient
ProxylessNAS [CZH19]†	2.08	5.7	4.0	gradient
PARSEC [CGF19]	2.81 ± 0.03	3.7	1	gradient
P-DARTS [CXW19]	2.50	3.4	0.3	gradient
PC-DARTS [XXZ20]	2.57 ± 0.07	3.6	0.1	gradient
SDARTS-ADV [CH20a]	2.61 ± 0.02	3.3	1.3	gradient
GAEA + PC-DARTS [LKB20]	2.50 ± 0.06	3.7	0.1	gradient
DrNAS	2.54 ± 0.03	4.0	$0.4^\ddagger$	gradient
DrNAS + progressive learning	2.46 ± 0.03	4.1	$0.6^\ddagger$	gradient

* Obtained without cutout augmentation.
† Obtained on a different space with PyramidNet [HKK17] as the backbone.
‡ Recorded on a single GTX 1080Ti GPU.

the expected validation loss controls the trace norm of the Hessian matrix. We refer the reader to the DrNAS paper [CWC21] for the detailed proof.

2.8.1.1 Experimental Results

Empirically, DrNAS significantly outperforms DARTS and comparable methods on CIFAR-10 (Table 2.3) and ImageNet (Table 2.4).

2.9 Beyond Differentiable NAS - Predictor-based Architecture Search

While previous sections mainly focus on analyzing and improving differentiable NAS methods, in this section we introduce an alternative paradigm of NAS called predictor-based architecture search. Unlike Differentiable NAS, Predictor-based methods do not rely on building weight-sharing supernets to estimate the perfor-

Table 2.4: Test error of architectures discovered by DrNAS on DARTS Space and ImageNet under mobile setting.

Architecture	**Test Error(%)**		**Params**	**Search Cost**	**Search**
	top-1	top-5	**(M)**	**(GPU days)**	**Method**
Inception-v1 [SLJ15]	30.1	10.1	6.6	-	manual
MobileNet [HZC17]	29.4	10.5	4.2	-	manual
ShuffleNet 2× (v1) [ZZL18]	26.4	10.2	~ 5	-	manual
ShuffleNet 2× (v2) [MZZ18]	25.1	-	~ 5	-	manual
NASNet-A [ZVS18]	26.0	8.4	5.3	2000	RL
AmoebaNet-C [RAH19]	24.3	7.6	6.4	3150	evolution
PNAS [LZN18]	25.8	8.1	5.1	225	SMBO
MnasNet-92 [TCP19]	25.2	8.0	4.4	-	RL
DARTS (2nd) [LSY19b]	26.7	8.7	4.7	1.0	gradient
SNAS (mild) [XZL18]	27.3	9.2	4.3	1.5	gradient
GDAS [DY19]	26.0	8.5	5.3	0.3	gradient
BayesNAS [ZYW19]	26.5	8.9	3.9	0.2	gradient
DSNAS [HXZ20]†	25.7	8.1	-	-	gradient
ProxylessNAS (GPU) [CZH19]†	24.9	7.5	7.1	8.3	gradient
PARSEC [CGF19]	26.0	8.4	5.6	1	gradient
P-DARTS (CIFAR-10) [CXW19]	24.4	7.4	4.9	0.3	gradient
P-DARTS (CIFAR-100) [CXW19]	24.7	7.5	5.1	0.3	gradient
PC-DARTS (CIFAR-10) [XXZ20]	25.1	7.8	5.3	0.1	gradient
PC-DARTS (ImageNet) [XXZ20]†	24.2	7.3	5.3	3.8	gradient
GAEA + PC-DARTS [LKB20]†	24.0	7.3	5.6	3.8	gradient
DrNAS†	24.2	7.3	5.2	3.9	gradient
DrNAS + progressive learning†	23.7	7.1	5.7	4.6	gradient

† The architecture is searched on ImageNet, otherwise it is searched on CIFAR-10 or CIFAR-100.

mance of child architectures. Instead, they build a surrogate predictor to infer the performance of child architectures; The predictor can be trained using a tiny selected subset of all architectures (usually in hundreds of architectures), and subsequently deployed for architecture evaluation the efficiently. This way, predictor-based methods are free from the inductive biases of weight-sharing differentiable NAS. However, search efficiency becomes a major bottleneck for predictor-based NAS: To label the training pool for the surrogate predictor, we now need to train and evaluate hundreds of architecture fully from scratch. To improve the search efficiency, previous works mainly focus on developing more sample-efficient predictors. We tackle this challenge from different perspective: pause and resume the training of poor architectures to save budget. The resulting framework, RANK-NOSH, reduces the search budget of the best predictor-based algorithm by 5 folds, while achieving competitive performance.

2.10 Predictor-based NAS

2.10.1 Framework

Starting from a pool of randomly selected architectures, previous methods iteratively conduct the following steps: 1) train and evaluate all the architectures in the pool fully; 2) fit a surrogate performance predictor; 3) use the predictor to propose new architectures and add them to the pool for the next round [DCA20, WNS19, YZA20]. Compared with previous RL and evolution-based NAS methods, using a performance predictor can reduce the number of networks evaluated from scratch. However, training all the architectures in the candidate pool fully is still extremely computationally expensive. Most complementary advances alone this line focus on developing better predictors that require a smaller training pool [DCA20, WNS19, YZA20], but the potential to further cut down the search cost by reducing the training length of individual architectures in the pool has not drawn much attention.

2.10.2 Improving the efficiency of predictor-based NAS with early termination and learning to rank

Inspired by successive halving [JT16], our key idea is that the learning process of poor architectures can be terminated early to avoid wasting budgets. However, it is non-trivial to integrate successive halving to predictor-based NAS formulations. Firstly, predictor-based algorithms iteratively add new architectures to the candidate pool [DCA20, WNS19, YZA20], whereas regular successive halving only removes underperforming candidates from the initial pool. Secondly, with successive halving, architectures in the pool will be trained for different number of epochs, so their validation accuracy are not directly comparable in a semantically meaningful way. Standard regression-based predictor fitting, which requires the exact validation accuracy for each architecture when fully trained, will be problematic in this setting.

To tackle those challenges in a unified way, we introduce RANK-NOSH, an efficient predictor-based framework with significantly improved search efficiency. RANK-NOSH consists of two parts. The first part is NOn-Uniform Successive Halving (NOSH), which describes a multi-level scheduling algorithm that allows adding new candidates and resuming terminated training process. It is non-uniform in the sense that NOSH maintains a pyramid-like candidate pool of architectures trained for various epochs without discarding any candidates. For the second part, we construct architecture pairs and use a pairwise ranking loss to train the performance predictor. The predictor is essentially a ranking network and can efficiently distill useful information from our candidate pool consisting of architectures trained for different epochs. Moreover, the proposed framework naturally integrates recently developed proxies that measure architecture performance without training [AMD21, CGW21, MTS20], which allows more architectures to be included in the candidate pool at no cost.

2.10.2.1 Experimental Results

On DARTS Space, RANK-NOSH achieves an average test error of 2.53% on CIFAR-10 with over 5x search budget reduction than previous SOTA predictor-based NAS method arch2vec, which obtains an average test error of 2.56%.

2.11 Limitations of differentiable architecture search

While Differentiable Architecture Search (DARTS) offers a promising framework for efficient neural architecture search, its practical application is significantly hampered by limitations in the search space. Currently, the search space in DARTS is defined by a set of high-level operations (e.g., convolution, skip connection, pooling). This restricts the candidate architectures to human-designed concepts, primarily CNNs. Consequently, recent innovations like transformers cannot emerge from this limited search space. This raises a crucial question: how can we define a search space that is both compact and generic enough to generate truly novel networks? Our key inspiration comes from modern deep learning libraries, where architectures are internally represented as computation graphs. By constructing a search space of computation graphs, we can represent a broader range of architectures. This approach transcends architecture design and extends to any machine learning module representable as a computation graph. In the next section, we delve into a concrete application of this idea: optimizer search.

CHAPTER 3

Generic Search Space

3.1 Problem settings of optimizer search

Motivated by a vision of democratizing machine learning, the central objective for automated machine learning (AutoML), such as automated architecture [ZL17, LSY19b, WCC20] / optimizer [BZV17, ADG16, ZCH21, LM16, CCC21, CHC17] / loss [LYL19] / augmentation search [LSY19a], lies in reducing the need for expert design on a diverse set of tasks. To achieve this goal, it is critical for AutoML systems to exhibit a high level of efficiency, so that they can be directly applied to a variety of tasks without consuming a humongous amount of computing resources. A widely successful example of such an effort is DARTS [LSY19b] in Neural Architecture Search (NAS), which reduces the search cost from thousands of GPU days of early RL-based algorithms to a single digit, enabling direct application of NAS systems to a wide range of tasks [CLQ20].

Inspired by the success of efficient NAS methods, we turn our attention to another important but much less studied area of AutoML - **Automated optimizer search, where an efficient, scalable and generalizable framework is still absent.** Optimizer search aims to automatically design a suitable update function that takes gradients as inputs and produces update directions for the optimizee's parameters. Pioneering work in this area, coined *Learning to Optimize (L2O)*, adopts a data-driven approach by replacing human-designed update rules with a learnable parametric function [ADG16, LM16, CHC17, CCC21]. However, parametric optimizers are fundamentally not scalable to large models or datasets, as inferring its parameters typically requires expensive meta-learning steps such as backpropagating through gradient descent [BZV17, ADG16, ZCH21]. Moreover, the learned optimizer often generalizes poorly to even minor variants of its training task (Figure 3.3) [ADG16, ZCH21]. Poor scalability and generalization prevent L2O from being served as a general-purpose optimizer search framework that can be **directly applied to tasks of interest**.

The aforementioned limitations of parametric optimizers bring our attention to another line of method that searches over the discrete space of non-parametric update functions, which generally exhibit the same level of scalability and generality as human-designed optimizers [BZV17, RLS20]. NOS-RL [BZV17] extends early RL-based NAS framework [ZL17] to optimizer search, proposing to learn a sequential controller to produce optimizer update rules according to a predefined pattern. However, NOS-RL is sample inefficient, requiring over 10k evaluations to find good candidates. More recently, AutoML-Zero [RLS20, CLH22] proposes to search over the vast space of computer codes for the entire ML pipeline (including the optimizer). The excessive generality of its search space makes it even more costly to run than RL-based method. The search cost of existing non-parametric optimizer search frameworks makes them computationally prohibitive not only for practitioners to apply but also for researchers to analyze.

With the goal of democratizing research and practical applications of automated optimizer design, we introduce the first efficient, scalable, and generalizable optimizer search framework that can be directly applied to a wide range of tasks. We observe that non-parametric update rules are essentially mathematical expressions, with an innate tree structure where nodes are elementary math operators and edges represent their I/Os. Consequently, generating an update rule can be viewed as progressively appending nodes to the expression tree until it is complete. Inspired by this observation, we re-imagine the optimizer search space as a super-tree of mathematical expressions. Each leaf node on the super-tree contains an optimizer, and the path towards it represents the generation process of that optimizer's underlying expression. With the tree-structured search space, optimizer search can be naturally formulated as a path-finding problem, allowing a wide range of well-established tree-traversal methods to be used as the search algorithm. We show that a simple adaptation of Monte Carlo Sampling [ENP19, SZS20], equipped with our proposed rejection sampling and equivalent-form detection, can already produce remarkable results on our search space within a fraction of budgets compared with NOS-RL ($\sim 1\%$).

We extensively evaluate the proposed framework on a diverse set of learning tasks: digit classification with MNISTNET [ADG16], image classification with ConvNet [BZV17], graph learning with (Cluster-)GAT [VCC17, CLS19], norm-bounded adversarial attack on robustly trained models [MMS17a, CH20b, CAS20], and BERT fine-tuning on NLP datasets [KT19, WDS20]. These tasks cover both constraint and unconstrained optimizations and span over a large variety of models and datasets. Despite the simplicity, the proposed framework is able to discover update rules that surpass human-designed optimizers and prior optimizer search methods, with a budget of only 128 evaluations. We hope the proposed framework could lower the barrier of entry to practical non-parametric optimizer search, thereby providing an entry point for researchers and practitioners from ML community and beyond to study and utilize automated optimizer search systems.

3.2 Efficient, scalable and generalizable framework for optimizer search

3.2.1 Optimizer design space

Notations and problem formulation Deep learning tasks are frequently expressed as optimizing a loss function $L(\cdot)$ defined over parameter domain $\theta \in \Theta$. The minimizer of L can thus be obtained by $\theta^* = \arg\min_{\theta\in\Theta} L(\theta)$. For differentiable functions, a standard optimizer typically takes the form of iterative gradient descent: $\theta_{t+1} = \theta_t - \gamma * \phi(\nabla_\theta L(\theta_t))$, where t is the current iteration, γ is the learning rate and ϕ denote the update function. Existing optimizers primarily differ in their design of update function ϕ; For example, vanilla gradient descent uses identity mapping $\phi(x) = x$ as the update function, whereas Adam adopts a momentum-based dynamic learning rate schema: $\phi(\nabla_\theta L(\theta_t)) = m(\nabla_\theta L(\theta_t))/\sqrt{m((\nabla_\theta L(\theta_t))^2)}$, where $m(\cdot)$ denotes the momentum function with an internal state.

The goal of optimizer search is to automatically find a suitable update function ϕ over some hypothesis space Φ. The hypothesis spaces used in prior work can be divided into two categories: non-parametric and parametric spaces. Most human-designed optimizers belong to the first category, where the update function ϕ is not trainable. Learnable optimizers, such as L2LGD2 [ADG16] and SymbolicL2O [ZCH21], fall into the second category. Our work mainly focuses on non-parametric optimizer search, with the goal of providing an efficient, scalable and generalizable optimizer search framework that can be directly apply to various tasks.

Optimizer update rules as expression trees The first step toward such a framework is to understand the structure of non-parametric optimizers. We realize that, fundamentally, optimizers are mathematical expressions consisting of elementary operators ($+$, $-$, $sign()$, $inputs$, e.t.c.). Math expressions have an inherent tree structure that preserves its order of execution, where nodes are operators and edges represent their I/Os. For instance, Diagram 3.1 shows the expression tree of Adam [KB14]:

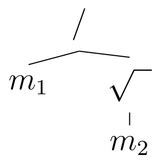


Diagram 3.1: Adam optimizer

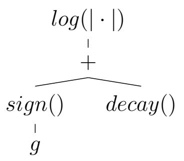


Diagram 3.2: Our discovered Optimizer for adversarial attack

where m_1 and m_2 denote the first and second order momentum (which can also be broken down into their own expression trees).

Therefore, the generation of an update rule, as a mathematical expression, can also be conducted via top-down node selection: Take Adam as an example, we first select division (/) as the root node. For its left child, we pick m_1, which is a leaf node and thus ends the branch. For the right child, we select $\sqrt{\ }$, and subsequently pick m_2 to follow it. At this point, there is no empty branches left, and we obtain the complete update rule for Adam.

A tree-structured search space Inspired by the completion process of update rules, we rearrange all expressions into a **super-tree**, where each leaf node contains an update rule and each path represents its completion process. The super-tree can be generated in a top-

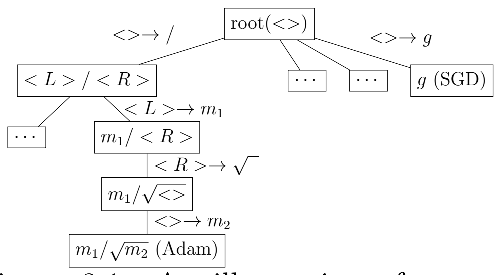


Figure 3.1: An illustration of traversing the super-tree to discover Adam and SGD.

down manner: Starting from the root
node with an empty update rule, we generate each of its child nodes by inserting a different operator into the update rule, and repeat this process for the generated nodes. Consequently, an optimizer can be sampled by traversing the super-tree until a leaf node is reached. Since the super-tree can grow infinitely deep, it is often desirable to restrict the tree to a predefined depth N, where only the paths that can be completed within depth N are included. Figure 3.1 provides an instantiation of our super-tree, where the paths leading to Adam and SGD optimizers are displayed as an example.

The benefit of arranging the optimizer space into a tree is two folds. Firstly, the tree-based search space is tight:

Proposition 3. *Define the length of an update rule as the number of operators it includes, then the above tree-based search space is* ***tight****: a super-tree with a maximum depth of N covers all update rules of length no greater than N.*

In a tight search space, all optimizers can be represented at the right level of complexity, allowing them to be visited by the search algorithm without exploring unnecessarily deep into the super-tree. Although tightness is a fairly obvious result for our space, it is not the case for the previous search space defined in NOS-RL, as we will explain later. Secondly, with our super-tree, optimizer search can be naturally formulated as a path-finding problem, allowing a variety of well-established tree traversal methods to be deployed as search algorithms.

Contents To concretize the content of the search space, we allow three types of operators in the optimizer update rule:

- a set of p_1 unary operators (e.g. $log(|\cdot|)$, $exp(\cdot)$, $\sqrt{|\cdot|}$, $sign(\cdot)$, $drop(\cdot)$)

- a set of p_2 binary operators (e.g. $+$, $-$, $\times$, $/$, $pow(\cdot, \cdot)$)
- a set of L leaf values (input operators) containing gradient-based terms (e.g. g, m_1), decays (e.g. *cosine_decay*), and constants (e.g. 1, 2)

This categorization of mathematical operators is not new, as it is also adopted in symbolic math solver [LC19] and NOS-RL [BZV17].

Comparison with NOS-RL's search space Although both NOS-RL [BZV17] and our framework use elementary math operators as building blocks for optimizers, they have little in common in terms of the arrangement of the search spaces. Optimizers in NOS-RL's search space are formed by a chain of predefined motifs: $b(u(I), u(I))$, where b, u, I denote binary, unary and input operators. Due to the fixed structure of such motifs, NOS-RL's search space is not tight: there exist many optimizers that take extra longer sequences to express, potentially lowering their chance of being discovered by the search algorithm. For instance, Diagram 3.2 shows an optimizer of length 5, but it takes $(10 - 1)$ nodes (two chained motifs) to represent it under NOS-RL's arrangement; Moreover, NOS-RL's search space also requires extra bypass operators (e.g. $u(x) = x$ and $b(x, y) = x$) to cover even human-design optimizers such as Adam and PGD, further increasing the complexity. In contrast, our representation of optimizers is directly inspired by the innate structure of its underlying mathematical expressions, resulting in a tight tree-based search space. In our search space, optimizer search can be naturally formulated as a form of top-down path-finding problem. In the next sections, we will detail our choice of algorithms for traversing the super-tree, as well as several techniques that leverage the characteristics of optimizer update rules to boost the sample efficiency.

3.2.2 Monte Carlo Sampling for tree traversal

We adopt a simple adaptation of Monte Carlo Sampling to tree traversal [Shr14, ENP19, SZS20] (MCT) as the search algorithm. The idea is to assign scores to the nodes in the super-tree (Figure 3.1), and use these scores to guide the tree traversal. We define the score of a node v as a Monte Carlo estimation over unrolling steps from v: If v is an internal node, we randomly generate a set of unrolled paths from v to the corresponding leaf nodes, and take the average score of the resulting optimizers as the score for v; If v is a leaf node, we set its score to 0 as it cannot be expanded. The search can thus be conducted as follows: 1). Starting from the root node $v^{(0)}$ at level 0, we generate all child nodes $\{v^{(1)}\}$ of $v^{(0)}$ by inserting each operator from the candidate pool to the update rule in $v^{(0)}$; 2). From there, we select the child node $v^{*(1)}$ with the highest MC score to expand, and move on to the next level; 3). The process is repeated until a predefined maximum search level is reached.

Directly applying the MCT algorithm to optimizer search would not perform well under limited search budgets, due to two unique characteristics of optimizer update rules that challenge the sample efficiency of the Monte Carlo estimates. Firstly, the majority of mathematical expressions, when deployed as optimizer update rules, perform poorly or even would not converge. This is usually not the case for other AutoML tasks such as neural architecture search, as most networks in the search space perform reasonably well. The large body of poor-performing optimizers not only consumes precious search budget, but also causes the MC estimation to be unstable. Secondly, there exists many mathematical redundancies in the expression space, for example: $sign(sign(sign(x)))$ can be reduced to $sign(x)$, and $\frac{m_1+\sqrt{m_2}}{\sqrt{m_2}}$ is equivalent to $\frac{m1}{\sqrt{m2}}+1$. Identifying and eliminating these redundancies would not only save budget, but also prevent the sampling distribution from biasing toward mathematically simple and shallow update rules. To address these issues and further boost the sample efficiency, we propose two sets of techniques - rejection sampling and equivalent-form detection. When combined with these techniques, the simple MCT algorithm becomes particularly effective for the optimizer search task. We will discuss them in detail in the following sections.

3.2.3 Rejection sampling

Eliminating poor optimizers with a train-free task-agnostic test Inspired by the characteristics of optimizer update rules, we develop a train-free task-agnostic test to eliminate poor optimizers without evaluating them. We propose a necessary condition for a valid optimizer: it must produce an acute angle with steepest descent direction (i.e. gradients). We check the validity of optimizers against this condition and only evaluate those that pass the test. For the test to be task-agnostic, we feed the optimizer with a batch of random Gaussian vectors in place of actual gradients. Formally, the descent test can be written as:

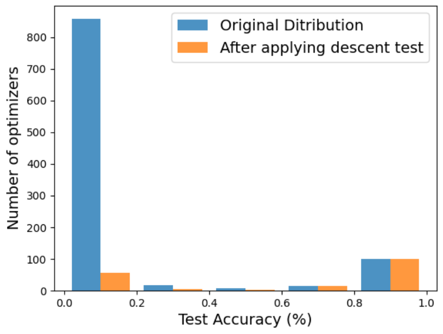


Figure 3.2: Performance distribution of optimizers after applying descent test, under $\lambda_d = 0.15$ and a batch size of 25.

$$E_{u \sim \mathcal{N}(0,1)}\left[\cos(\phi(\nabla_\theta \mathcal{L}), \nabla_\theta \mathcal{L})\right] > \lambda_d$$

where λ_d is a predefined threshold. Although our descent test is by-no-mean comprehensive, it can effectively rule out a large chunk of poor optimizers with negligible false-negative rates, as demonstrated in Figure 3.2.

Reducing the variance of MC estimates via score thresholding After applying the descent test, there still remains a non-negligible portion of poor optimizers. When sampled during the unrolling step, these optimizers would drastically lower the Monte Carlo score of the stem node, causing the MC estimation to exhibit high variance and thus become unreliable. This adverse effect is especially severe under the efficient setting when the sample size is small. Therefore, we propose to simply reject candidates with scores lower than a predefined threshold, thereby removing them from the MC scores of the corresponding stem nodes.

3.2.4 Detecting and handling redundancies in mathematically equivalent forms

On-the-fly constraint tree-traversal for redundant path pruning One benefit of formulating the search problem as top-down path-finding is that we can easily apply constraints on-the-fly to eliminate undesirable branches - those that lead to mathematically redundant expressions in our case. We identify three main categories of such constraints:

- child operator that nullifies its parent's operator. e.g. $-(-x) = x$, $ln(e^x) = x$
- child operator that is redundant under its parent. e.g. $clip(clip(x)) = clip(x)$
- sequence of operators that reduces to a constant in the search space. e.g. $\sqrt{|sign(x)|} = 1$

Enforcing these constraints during the traversal can effectively trim down the search tree, allowing the algorithm to explore branches that lead to more diverse and complex expressions.

Hashing mathematically equivalent expressions Besides enforcing constraints during the traversal, it is also important to detect mathematically equivalent optimizers to avoid duplicated evaluations. One can always apply off-the-shelf symbolic solvers to identify the equivalence of two expressions, ϕ and ϕ', by checking if $(\phi - \phi')$ can be reduced to 0. However, it could become extremely slow as the pool of evaluated optimizers $\{\phi_i\}_1^N$ gets larger and larger, since we need to solve N pair of equations every time a new update rule is sampled. Instead, we apply hashing to efficiently query the evaluated candidate pool for mathematically equivalent optimizers. Concretely, we assign each optimizer a hash code, obtained by feeding a fixed probing vector as input to the optimizer and recording its output. The probing vector is pre-sampled from Gaussian distribution. When a newly sampled optimizer arrives, we only need to compare its code with the hash table to check the existence of its equivalent form. Empirically, it is much faster to run the proposed hashing-based checker than symbolic solvers.

3.3 Empirical evaluations on a diverse set of tasks

We extensively evaluate the proposed framework on a suite of tasks, covering a variety of models and datasets. On standard benchmark tasks for optimizer search, our method is able to discover optimizers that outperform its human-designed and automatically searched counterparts. In addition, we also show that the proposed framework enables automated optimizer design for many other popular learning tasks, such as adversarial attack, GNN training, and BERT finetuning.

3.3.1 General setting

MCT algorithm Across all experiments, we limit the maximum level of MCT traversal to 4, and set the number of Monte Carlo samples to 32 (a multiple of 8 for parallelism on 8-GPU servers) for each level. This amounts to a fixed total budget of 128 evaluations. The maximum depth for the super-tree is set to 10, which already covers many top-performing optimizers for various tasks. We use a similar set of elementary operations as NOS-RL to build the optimizers, with only minor adjustments for some tasks.

Optimizer evaluation We follow the default settings and hyperparameters for each task, and only swap out the optimizer; This potentially puts our algorithm at a disadvantage, as the hyperparameters are usually tuned around the default optimizers. Before optimizer evaluation, we perform grid search on a small proxy task (fewer steps) to find a proper learning rate. During the grid search, we also aggressively terminate optimizers if their performance falls under a certain threshold. Since early stopped optimizers consume fewer resources than a full evaluation, we do not count them into the budget (number of evaluations).

3.3.2 Hand-written digit classification

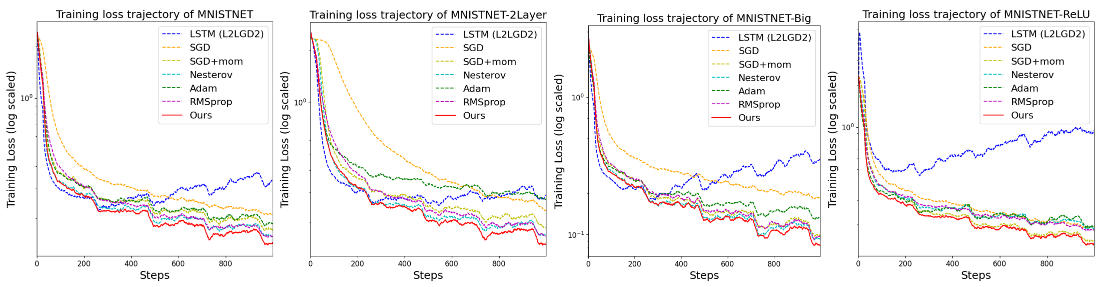


Figure 3.3: Training loss trajectory on hand-written digit classification task (log scaled). Each optimizer is evaluated for 4 random seeds. Our method is marked in red.

We first compare our method with the LSTM-based optimizer (L2LGD2) on hand-written digit classification. Following L2LGD2 [ADG16], the goal is to minimize the cumulative training loss of a single-hidden-layer MLP with Sigmoid activation (MNISTNET) on the MNIST dataset; The search is conducted on MNISTNET for 100 steps with a batch size of 128, and the discovered optimizers are subsequently transferred to three variants of MNISTNET with different activations (MNISTNET-ReLU), number of hidden layers (MNISTNET-2Layer), and dimensions (MNISTNET-Big). Under this setting, our method finishes in 0.92h on RTX 2080ti, much faster than L2LGD2 (2.62h).

As shown in Figure 3.3, our discovered optimizer (e.g. $m_1 + RMSprop * exp(Adam)$) achieves the lowest training loss under both direct search and transfer settings. Notable, the LSTM-based parametric update function indeed converges faster when the number of steps is close to the search phase (black-dotted vertical line on Figure 3.3). However, it extrapolates poorly to longer trajectories. As the training proceeds, all other non-parametric optimizers eventually catch-up, achieving much lower training loss. Moreover, LSTM-based optimizer also generalizes poorly to other model variants (most noticeably MNISTNET-ReLU), revealing its tendency to overfit the search task.

3.3.3 Image classification with ConvNet

We proceed to evaluate our method on the CIFAR-10 classification task proposed in NOS-RL [BZV17]. The model of choice is a 2-layer ConvNet. Each layer of this network contains a 32-filter 3x3 convolution with ReLU activation and batch normalization. Following NOS-RL's setting, for every optimizer, the best learning rate is searched over a grid of $\{1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}, 1e^{-1}, 1\}$ with 1 epoch of training, and the discovered learning rate is subsequently used to train the model

Table 3.1: Performance of automated search algorithms on CIFAR-10.

Optimizer	**Test Accuracy (%)**	**Search Method**	**Search Budget (#evaluations)**
SGD	70.99% ± 2.12	manual	-
SGD + Momentum	74.12% ± 0.44	manual	-
Nesterov	74.15% ± 0.52	manual	-
Adam	73.42% ± 0.56	manual	-
RMSprop	71.42% ± 1.42	manual	-
PowSign-ld	75.48% ± 0.45	RL on hand-crafted patterns	¿10,000
PowSign-cd	76.21% ± 0.16	RL on hand-crafted patterns	¿10,000
AddSign-ld	75.54% ± 0.39	RL on hand-crafted pattern space	¿10,000
AddSign-cd	76.07% ± 0.59	RL on hand-crafted pattern space	¿10,000
Ours	**77.02% ± 0.19**	MCT on super-tree space	**128**

for a longer period of time (5 epochs). Since NOS-RL's implementation is not open-sourced, we reproduce and compare with the two families of discovered optimizers described in NOS-RL paper: AddSign and PowSign.

The results are summarized in Table 3.1. For NOS-RL, we display the performance of the top 4 variants of PowSign and AddSign, which are obtained after training the controller for over 10k evaluations (Figure 4 in the NOS-RL paper [BZV17]). With only a fraction ($\sim$1%) of the search budget, our framework is able to discover optimizers (e.g. $cos\_decay * drop_{0.1}(g)/linear\_decay$) that reach a test accuracy of 77.02%, topping both PowSign and AddSign optimizers and also human-designed ones by a sizable margin. The sheer reduction in search cost and the improvement in search performance evince the efficiency and effectiveness of the proposed framework for discovering better optimizers.

3.3.4 Adversarial attack

Next, we apply our framework to discover optimizers for constraint optimization. We select adversarial attack, which aims at finding norm-bounded perturbations in the input space that alter the model's predictions. The de facto optimizer used in adversarial attack is Projected Gradient Descent (PGD) [MMS17a]. We consider the most popular l_∞-norm setting. For l_∞-norm bounded attack, PGD takes the form of: $x = Proj_{B^\infty_\epsilon(x_o)}(x + \gamma sign(\nabla_x L(x)))$, where $B^\infty_\epsilon(x_o)$ represents a ϵ ball around the original image x_o w.r.t. l_∞-norm. The models of choice come from the AutoAttack library [CH20b], which holds a leaderboard of top defense methods. Following their settings, we set $\epsilon = 8/255$, and run each optimizer once for 100 steps on every image from the test split [CH20b].

Table 3.2: Attack success rate of different optimizers on top defense methods on CIFAR-10.

Defense Models	**PGD**	**APGD**	**Ours**
Carmon2019 (WRN-28-10) [CRS19]	37.83%	38.22%	**38.35%**
Gowal2020‡ (WRN-70-16) [GQU20]	31.10%	**32.00%**	**32.00%**
Gowal2020‡ (WRN-34-20) [GQU20]	40.05%	40.46%	**40.50%**
Gowal2020‡ (WRN-28-10) [GQU20]	33.65%	34.33%	**34.34%**
Sehwag2020‡ (WRN-28-10) [SWM20]	40.00%	40.43%	**40.46%**
Wu2020‡ (WRN-28-10) [WXW20]	36.41%	36.70%	**36.78%**
Wang2020‡ (WRN-28-10) [WZY19]	37.78%	38.16%	**38.27%**
Engstrom2019 (RN-50) [EIS19]	47.76%	48.25%	**48.32%**
Wong2020Fast (RN-18) [WRK20]	53.69%	54.11%	**54.19%**

‡ Methods that explore extra data during robust training.

On this task, we mainly search for the update rule inside the projection operator (e.g. $sign()$ for PGD). The search is conducted on the pretrained Carmon2019 model [CRS19], and the proposed optimizer is subsequently evaluated on other top defense methods for WideResNet (WRN-¡depth¿-¡width¿) and ResNet (RN-¡depth¿). As shown in Table 3.2, our discovered optimizer consistently outperforms PGD by a sizable margin. Surprisingly, we found that the algorithm tends to pick $log(|\cdot|)$ rather than $sign(\cdot)$ as the first operator, resulting in many log-based optimizers that surpass sign-based PGD (e.g. $log(|cos\_decay + sign(G)|)$).

In addition to PGD, we also compare our log-based optimizer with the best handcrafted and tuned optimizer for adversarial attack: Adaptive PGD (APGD) [CH20b]; The design of APGD is packed with domain expertise: it combines a well-tuned momentum update rule with a conditional learning rate decay based on a handcrafted schedule and sophisticated decay conditions. However, the performance of our automatically discovered optimizer rivals APGD across various defense methods, despite of having a much simpler form. This result demonstrates the potential of applying our framework to reduce the need of human expertise in designing optimizers for diverse tasks.

3.3.5 Node classification on graphs

We next test our framework for optimizing graph neural networks to classify nodes on graphs. The model of interest is Graph Attention Network (GAT) [VCC17], one of the most widely used architectures in graph learning tasks. We compare our method against Adam [KB14] - the standard optimizer for optimizing GATs - on five commonly used graph datasets: OGBN-Product [HFZ20], Cora, Citeseer, PubMed, and PPI. Among them, OGBN-Product is the largest in scale, consisting of 2,449,029 nodes. Since standard GATs cannot scale to this dataset, we instead adopt an adaptation of cluster-GCN [CLS19] to GAT as the testbed, termed Cluster-GAT. Cluster-GAT trains standard GAT on smaller partitions of the original graph, thereby allowing the model to be applied to large-scale graphs.

Table 3.3: Performance of our discovered optimizers against Adam on GATs on five commonly used Graph datasets of diverse size. Results that use the same GAT implementations are grouped together.

Dataset	**Adam**	**Ours**
Products	77.49% ± 0.56[†]	**80.15% ± 0.16**
Cora	84.72% ± 0.32	**85.20% ± 0.19**
Citeseer	71.70% ± 1.03	**73.10% ± 0.43**
PubMed	78.20% ± 0.22	**79.25% ± 0.70**
PPI	97.53% ± 0.45[‡]	**98.13% ± 0.10**[‡]

[†] Our reproduced accuracy using ogbn-leaderboard's implementation is lower than the displayed number (79.23% ± 0.78).

[‡] F1 Score

The results are summarized in Table 3.3. On all datasets, our search algorithm is able to discover optimizers that outperform Adam. An interesting observation is that the top-performing optimizers discovered for this task almost always contain $sign(\cdot)$ operators (e.g. $linear\_decay * (sign(m_1) - RMSprop)$), revealing the potential of adopting sign-based optimizers to improve the training of graph neural networks.

3.3.6 BERT fine-tuning on NLP datasets

We also evaluate the proposed framework on BERT finetuning task on GLUE benchmark [WSM18]. For this task, we follow all configurations of the HuggingFace implementations: we finetune a pretrained BERT (base cased) model for 3 epochs on Cola, STS-B and RTE dataset, and 5 epochs on MRPC and WNLI dataset. The batch size is set to 32. We compare our discovered optimizers with the default AdamW. As shown in Table 3.4, our automatically discovered optimizers (e.g. $drop_{0.1}(clip_{0.003}(m_1 - \sqrt{|drop_{0.1}(g^3)|} * sign(m_1))))$ outperform AdamW on all datasets.

Table 3.4: Performance of our discovered optimizers for BERT fine-tuning on GLUE tasks.

Dataset	**AdamW**	**Ours**
Cola	$59.56 \pm 2.04^\star$	$\mathbf{60.89 \pm 1.33}^\star$
MRPC	$82.84 \pm 0.57^\ddagger$	$\mathbf{86.64 \pm 0.94}^\ddagger$
STS-B	$87.80 \pm 1.14^\dagger$	$\mathbf{88.91 \pm 0.30}^\dagger$
RTE	$65.97 \pm 1.56^\ddagger$	$\mathbf{68.50 \pm 1.93}^\ddagger$
WNLI	$53.17 \pm 5.49^\ddagger$	$\mathbf{56.34 \pm 0.00}^\ddagger$

* Mathews Correlation.

† Spearman Correlation.

‡ Accuracy (%).

3.4 Conclusion

Despite the recent advancement of practical AutoML systems in automatizing the design of architectures, data augmentation policies, and hyperparameters, progress in automated discovery of optimizers is still inadequate due to the limitations of prior methods in terms of 1). efficiency, 2). generalization, and 3). scalability. In this paper, we introduce the first optimizer search framework that meets all these criteria, allowing it to be directly applied to the tasks of interest. The proposed framework demonstrates promising results across a variety of tasks, from image classification, adversarial attack, to graph learning and BERT fine-tuning. Our method by-no-mean solves the optimizer search problem, as there is plenty of room for improvement on the algorithm and search space; Rather, our goal is to open up a new possibility for future development in non-parametric optimizer search methods. We hope the proposed framework could democratize research and applications of automated optimizer search, and stimulate interest among researchers and practitioners.

Part II

Inference: From Model Selection to Prompt Optimization

CHAPTER 4

When Model Selection Becomes Prompt Optimization

4.1 Introduction

From Bayesian perspective, architecture and optimizer search performs model selection: picking the best pretrained model for a given set of tasks. We show that for Large Language Models, it transforms into prompt optimization. Our key insight is that LLMs encompass a variety of powerful, conditional probabilistic sub-models. These models share a unified parametric architecture with the unconditional parent LLM (Super Model), yet distinctive defined by their respective prompts. Therefore, crafting prompts (by either Human or meta-LLMs) for LLM is equivalent to searching over the hypothesis space spanned by these submodels. The next section provides a theoretical justification for this connection.

4.2 Connection between model selection and prompting optimization

LLMs pretrained on the next-token prediction task model the following joint distribution of a sequence of tokens $\{w_t\}_{t=1}^T$

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, w_{t-2}, \ldots, 1) = f_\theta(w_t \mid w_1, w_2, \ldots, w_{t-1}),$$

where the conditional probabilities are parameterized by an auto-regressive model $f(\cdot;\theta)$ (e.g. Transformer) and each word w_t is predicted given all the preceding tokens. The pretraining objective minimizes the following negative log-likelihood:

$$\min_{\theta} \mathcal{L}(\theta) = -\sum\nolimits_{t=1}^{T} \log f_\theta(w_t \mid w_{t-1}, \ldots, w_1). \tag{4.1}$$

A key observation from Eq. equation 4.1 is that the training process optimizes a "SuperNet" of conditional probabilistic models (CPM), each defined by an instruction s: $f_{s,\theta}(y|x) = f_\theta(y \mid x, s)$, where x is the input and s is the instruction for a particular task. Therefore, with a fixed LLM, the set of natural language prompts, denoted as $\mathcal{S}$, provides a massive set of submodels for the user to select from. For a given dataset $\{(x_i, y_i)\}_{i=1}^{n}$, finding the best prompt to minimize the empirical loss,

$$\min_{s \in \mathcal{S}} \sum_{i=1}^{n} \mathcal{L}((f_{s,\theta}(y_i \mid x_i))),$$

which essentially selects the best submodels from the pretrained "SuperNet".

CHAPTER 5

Prompt Optimization for (Multimodal) Large Language Models

5.1 Problem setting

Learning interpretable predictive models from annotated data remains a key challenge in human-centric AI. Given input-output pairs $\{(x_i, y_i)\}$, the objective is to learn a function $f : x \to y$ that not only fits the data accurately but is also interpretable. tIn this context, a strong form of "interpretable" means that individuals with no prior domain knowledge can understand and apply the decision rules demonstrated by f, facilitating *the transfer of knowledge from AI to humans*. This is crucial not only for enhancing the transparency of AI systems but also for enabling humans to learn from these models, empowering various human-in-the-loop applications such as scientific discovery, material synthesis, and automatic data annotation [PMS16].

Consider an exemplar task of classifying species in Palworld [Pai24] - a newly released Pokemon-style game - based on a few image-label pairs, as illustrated in Figure 5.1. The ultimate goal is that even humans unfamiliar with Palworld can replicate AI's decisions by following the same predictive rules after examining the model trained on the data. This task effectively represents the challenge of extracting interpretable knowledge, such as species characteristics, from data. The algorithm we propose in this paper learns a model following the decision rule illustrated in Figure 5.1, which is designed to be easily understood and reproduced by humans. In essence, this problem can be viewed as discovering interpretable knowledge (e.g., the properties of a species in Palworld) from the data.

Despite extensive research, the problem of developing a fully interpretable predictive model has not been fully addressed. Traditional methods often face a trade-off between expressiveness and interpretability: Deep neural networks, for instance, are powerful yet operate as "black boxes". Although post-hoc explanation methods attempt to make these models more transparent by identifying influential features [ZCA17, PDS18, DG17, SGK17, STY17, ACO17], they do not clarify the underlying decision-making processes and have no control over the learning process. Directly learning interpretable models like (locally) linear [RSG16], tree-based [Lun17] often falls short in expressiveness, especially with complex inputs like images.

To address this challenge, **Neurosymbolic Programs (NSPs)** [PMS16, SZS20, CZ21, NVS21] offer a promising solution by modeling the decision rule as a program incorporating both symbolic operations and neural network modules. Despite this, the inherent trade-off between expressiveness and interpretability persists. While the integration of neural modules enhances expressiveness, it also compromises the program's overall interpretability. Additionally, designing effective symbolic operators requires significant expertise and is critical for the performance of the resulting program, necessitating careful customization for each specific dataset [PMS16, SZS20, CZ21].

Is it possible to harness the power of neural networks within Neurosymbolic Programs without compromising interpretability? This paper presents an affirmative answer. Our key insight is that (Multimodal) LLMs encompass a variety of powerful, conditional probabilistic sub-models defined by various prompts; and the learning of those models can be achieved via prompt optimization techniques. This yields an infinite set of neural network-based operations that are inherently interpretable and can serve as fundamental "learnable" building blocks within Neurosymbolic Programs.

Building on this insight, we introduce a novel framework termed **LLM-Symbolic Programs (LSPs)**, defined and learned through LLMs. Our approach leverages a minimal Domain-Specific Language (DSL) set with only two operators: prompted-LLM and conditional branching, yielding a classic decision-making process structured as trees. We then propose a learning algorithm to incrementally learn the tree using LLMs with prompt optimization. To thoroughly evaluate the efficacy of LSPs, we construct the **Interpretable-Learning-Benchmark** of diverse predictive tasks, containing both synthetic and real-world data across vision and text modalities. Our empirical findings show that LSPs surpass the accuracy of both traditional XAI methods and LLMs prompted with automatically learned instructions, all while maintaining human interpretability. These results highlight the potential of LSPs to significantly enhance the performance and utility of Multimodal LLMs in various applications.

5.2 Related Work

Prompt Optimization The essence of utilizing a generative language model lies in crafting effective prompts. Recent advancements have aimed to automate this process, reducing the need for human effort through prompt optimization [SRL20, ZMH22]. While pioneering efforts were mainly directed towards various discrete optimization algorithms [SRL20, DWH22, ZWZ22], it has been noted that advanced LLMs can revise prompts similarly to human engineers [ZMH22, PIL23]. Since these initial efforts, a significant body of research has emerged, exploring various search algorithms including Monte Carlo Sampling [ZMH22], beam search [PIL23], evolutionary search [YWL23a, FBM23, XCD22, GWG23a, HSY23], and tree search [WLW23]. However, existing methods often treat the prompt as a single entity without explicit structure. From this perspective, prompt optimization methods can be seen as simplified instances of LSPs, where the program consists solely of one LLM module. While this simplification has shown promising results, as task complexity increases, the explicit structuring within LSPs allows them to encode knowledge from data. This provides substantial advantages over conventional prompt optimization methods.

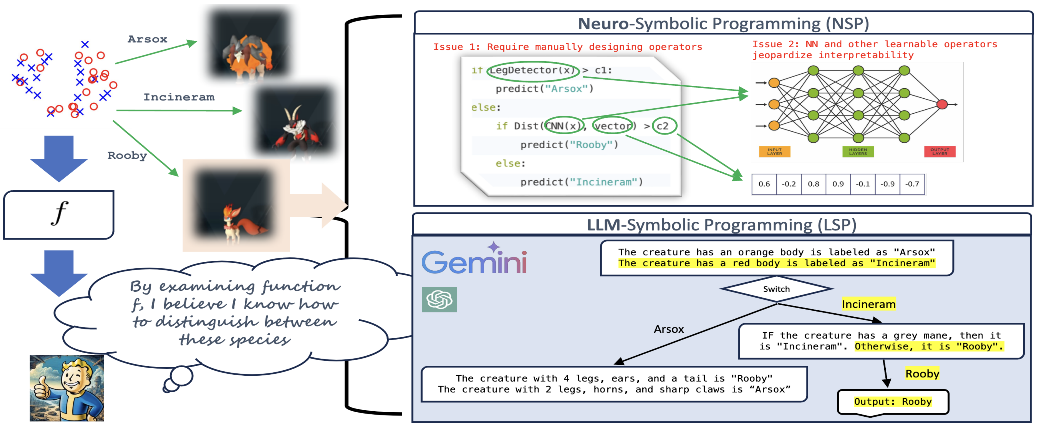
5.3 IL-Bench: 1st Interpretable-Learning Benchmark for (M)LLMs

To address the lack of suitable benchmarks for evaluating the interpretable learning capabilities of (M)LLMs, we introduce the **I**nterpretable-**Learning** Benchmark (IL-Bench). This new benchmark comprises a series of challenging tasks that are not solvable through zero-shot methods by even the most advanced (M)LLMs, such as GPT-4 and Gemini-1.5. IL-Bench includes 16 new symbolic and real-context tasks unseen to the current model lineup. These tasks range across vision and language modalities, providing a comprehensive and extensible evaluation framework. Below, we provide a high-level summary of the key data curation methods.

Symbolic tasks Drawing inspiration from language-independent IQ tests, we generate set of synthetic datasets to evaluate the interpretable learning capabilities of the models. These datasets utilize symbols to denote input variables and their values; The input values are randomly assigned, and mapped to their labels based on a predefined set of rules. We also vary the number of variables, values, and labels to generate datasets of increasing complexity. These symbolic tasks enjoy several key benefits: 1. **Known oracle rules**, enabling precise evaluation of learning ability. 2. **Context independence**, forcing the models to depend solely on learned rules, without relying on external context. 3. **Scalability**, allowing for the automated creation of an unlimited number of tasks with arbitrary difficulty levels.

Textual classification tasks: converting vision dataset to text inputs To evaluate model proficiency in intricate real-world scenarios, we utilize Fine-Grained Visual Classification (FGVC) datasets [MRK13, WBW11, KP20, NZ08, VBF15], such as CUB commonly used in XAI research. These datasets comprise of objects within narrowly-defined, visually-similar categories that are particularly challenging for the model to distinguish. To adapt these visual datasets for textual evaluation, we convert them into text-based datasets using a captioning model. In order for the task to be well-defined, the generated caption must cover all visual features required for classification, which are usually very subtle for FGVC datasets (e.g. the particular shape of a bird's beak). To ensure the captions capture all essential visual features, we also provide contrastive examples to the captioner (details in Appendix). The class names (e.g. Sea_Albatross) are also anonymized by symbols (e.g., class_1) to prevent the model from using label names to "shortcut" the prediction process. Empirical results indicate that the performance of existing text-based LLMs approximates that of random guessing in zero-shot setting.

Visual classification Tasks: distinguishing novel visual concepts Due to the extensive coverage of (M)LLM training data, evaluating models in a multi-


Arsox
Incineram
Rooby
f
By examining function f, I believe I know how to distinguish between these species
Neuro-Symbolic Programming (NSP)
Issue 1: Require manually designing operators
Issue 2: NN and other learnable operators jeopardize interpretability
INPUT LAYER
HIDDEN LAYERS
OUTPUT LAYER
0.6 -0.2 0.8 0.9 -0.1 -0.9 -0.7
LLM-Symbolic Programming (LSP)
Gemini
Switch
Incineram
Arsox
Rooby

```
if LegDetector(x) > c1:
    predict("Arsox")
else:
    if Dist(CNN(x), vector) > c2
        predict("Rooby")
    else:
        predict("Incineram")
```

```
The creature has an orange body is labeled as "Arsox"
The creature has a red body is labeled as "Incineram"
```

```
IF the creature has a grey mane, then it
is "Incineram". Otherwise, it is "Rooby".
```

```
The creature with 4 legs, ears, and a tail is "Rooby"
The creature with 2 legs, horns, and sharp claws is “Arsox”
```

```
Output: Rooby
```

Figure 5.1: **Illustration of LLM-Symbolic vs. Neuro-Symbolic Program on interpretable learning task.** The goal is to develop a model that allows humans with no prior knowledge to replicate AI's decisions by following the same rules as the model. While **NSP (Top right)** offers a certain level of interpretability, it heavily relies on manually designing operators, and the inclusion of neural operators often reduces interpretability. In contrast, **LSP (Bottom right)** generates fully interpretable programs with the help of versatile LLM modules.

modal setting presents a unique challenge. Despite our best efforts, all existing image classification datasets we tested were already seen by at least one (M)LLM, which can predict labels in a zero-shot manner. To address this, we curate seven new datasets using screenshots from "Palworld," a recently released regional game featuring various creature species similar to Pokémon. As this game was released after the knowledge cut-off dates of the tested (M)LLMs, the models lack prior information about these creatures, requiring them to rely solely on the knowledge extracted from the dataset for predictions.

5.4 Interpretable Learning with LLM-Symbolic Programming

This section explains our proposed framework: LLM-Symbolic Programs. Section 5.4.1 reviews Neurosymbolic Learning method. Section 5.4.2 discusses utilizing LLM to implement interpretable programs, including the Domain Specific

Language (Section 5.4.2.1) and learning algorithm (Section 5.4.2.2).

5.4.1 Preliminaries on classical Neurosymbolic Learning

NeuroSymbolic Programming (NSP) [PMS16, SZS20, CZ21, FH17] represents an innovative method for combining classical symbolic learning with contemporary neural networks, with the goal of building expressive and interpretable models. NSP often consists of two main components: (1) a **Domain Specific Language (DSL)** that specifies available operations of the program (akin to a "search space") and (2) a **learning algorithm** for finding the best program. The resulting programs are structured, neuro-symbolic terms that follow the syntax specified by the DSL.

Domain-Specific Language (DSL) DSL in NSPs comprises manually defined operators, including interpretable symbolic (e.g. if-then-else) and expressive neural components (e.g. cnn(x, θ)). These operators can be chained to construct various tree-structured programs, a.k.a. computation graphs. equation 5.1 presents an example DSL used to construct the program for predicting the creature species in Figure 5.1. Here, x and c represents inputs and constants, and α denotes a sub-program:

$$\alpha ::= x \mid c \mid \mathrm{Add}(\alpha_1, \alpha_2) \mid \mathrm{Mul}(\alpha_1, \alpha_2) \mid \mathrm{If}\ \alpha_1\ \mathrm{Then}\ \alpha_2\ \mathrm{Else}\ \alpha_3 \mid \mathrm{cnn}(x, \theta) \mid \mathrm{Dist}(\alpha_1, \alpha_2). \tag{5.1}$$

Co-optimization of program structure and learnable parameters In NSPs, the construction of a program involves solving a combinatorial optimization problem for both the program structure and the parameters of its learnable operators (e.g. neural components). As the number of DSL operators increases, the complexity of this task grows exponentially. To make the search process more tractable, existing research employs various approximation techniques to efficiently identify viable candidates, including greedy tree search [SZS20], continuous relaxation [CZ21], distillation [FH17] and meta-learning [PMS16].

Limitations While the integration of symbolic and neural components in NSPs represents a promising innovation, the incorporating of neural modules inevitably introduces black-box components and makes the program non-interpretable. Researchers have attempted to address this issue through two primary approaches: restricting the DSL to only interpretable operators [SZS20, CZ21], or employing prototype learning to derive relatively interpretable neural modules [NVS21, MXQ19, NJP21]. However, the DSL approach is not automatic, heavily relies on domain expertise, and potentially overlooking crucial information not identified by experts; Conversely, prototype learning aims to represent the concept of each neural module by a set of representative samples, which is not guaranteed to success.

5.4.2 LLM-Symbolic Programs

This section explores how LLMs can effectively be utilized to implement NSPs' modules that are expressive, interpretable, and straightforward to learn with LLMs.

5.4.2.1 Domain-Specific Language of LSPs

Traditional NSPs require manually designing a comprehensive DSL. However, with LLM's ability to represent a wide range of functions via different prompts, we can significantly streamline the grammar required to build expressive and interpretable models. Specifically, for predictive models, we can build powerful LSPs from a minimalist DSL with only three components: the input, conditional branching, and LLM module:

$$\alpha ::= x \mid \text{switch}(\{\alpha == y_i : \alpha_i\}_{i=1}^{k}) \mid \text{LLM}(x, s). \tag{5.2}$$

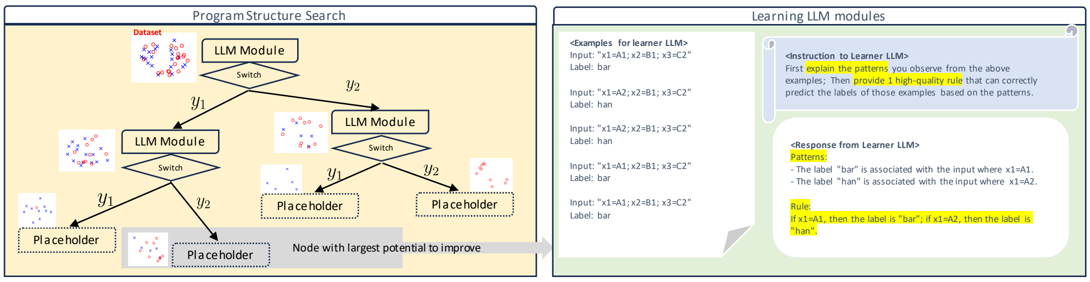
Here, **input** x represents the input data (text, image, etc); the **conditional branching** $\text{switch}(\{y_i : \alpha_i\}_{i=1}^{k})$ forms the backbone of the program structure. Each switch can be viewed as a node in a decision tree tree with k branches. It will branch to α_i if the sub-program α predicts y_i. **The LLM Module** $\text{LLM}(x, s)$ serves as the inference engines. It means to prompting LLM to make a prediction on input x under the instruction s.

Figure 5.1 (Bottom Right) shows an example program generated from above DSL. During inference time, given a test query, we traverse the tree-structured program in a top-down manner, assigning data to specific child node based on the parent node's predictions, until the leaf node is reached and the final response is returned.

5.4.2.2 Learning algorithm

After defining the search space for program construction, we proceed to describe the algorithm used to identify the optimal program. Similar to Neuro-Symbolic Programming (NSP), our approach involves optimizing two key components:

- **(1) LLM module optimization**: Generating the rules from data for each LLM module.
- **(2) Program structure search**: Determining how to expand the program tree.



<Examples for learner LLM>
Input: "x1=A1; x2=B1; x3=C2"
Label: bar

Input: "x1=A2; x2=B1; x3=C2"
Label: han

Input: "x1=A2; x2=B1; x3=C2"
Label: han

Input: "x1=A1; x2=B1; x3=C2"
Label: bar

Input: "x1=A1; x2=B1; x3=C2"
Label: bar

<Instruction to Learner LLM>
First explain the patterns you observe from the above examples; Then provide 1 high-quality rule that can correctly predict the labels of those examples based on the patterns.

<Response from Learner LLM>
Patterns:
- The label "bar" is associated with the input where x1=A1.
- The label "han" is associated with the input where x1=A2.

Rule:
If x1=A1, then the label is "bar"; if x1=A2, then the label is "han".

Figure 5.2: **Learning Algorithm for LSPs.** The learning algorithm for LSPs contains two parts: **(1) program structure search (Left):** This process is akin to constructing a traditional decision tree. Starting from the root, the algorithm traverses down the tree, iteratively splitting the training dataset based on the current node's predictions and expanding the leaf node with the highest prediction errors. **(2) LLM module optimization (Right):** Here, a learner LLM is instructed to summarize rules based on the observed data at its node.

Figure 5.2 illustrates the entire search process. The following sections will describe these two components respectively.

LLM modules optimization via summarizing predictive rules In Large Symbolic Programs (LSPs), each LLM module is responsible for making decisions on its designated data subset. While traditional NSPs optimize neural modules through empirical risk minimization, LSPs can derive predictive rules directly from observed data, a method we termed **RuleSum**. To achieve this, we leverage the LLM's powerful summarization capabilities [AFL23, GLD22, ZLD24, PD23], and instruct a learner LLM to observe patterns from the data samples and summarize them into concrete rules. The process is visualized in Figure 5.2 (right).

Program Structure Search LSP produces a tree-structured program where each path represents a complete decision-making process. To discover the optimal program, we employ a top-down tree traversal approach to expand the tree from scratch. Starting from the root node of an empty program with the entire training dataset:

- **Step 1:** Add an LLM(x, s) module to the root node.

- **Step 2:** Optimize LLM(x, s) using the **RuleSum** algorithm.
- **Step 3:** Create child nodes for the root by adding a switch operator to the program.
- **Step 4:** Assign training data to child nodes based on LLM(x, s)'s predictions.
- **Step 5:** Move to the highest-scoring child node, and repeat Steps 1–4 until max_iter is reached.

In essence, this search algorithm uses a divide-and-conquer strategy: it progressively partitions the training dataset into sub-branches based on the parent node's predictions, enabling the child LLM modules to further refine the prediction. This approach simplifies the learning process for each LLM module and makes the overall system more error-tolerant: the **RuleSum** algorithm only needs to derive rules for a subset of the data, and any inaccuracies can be corrected by subsequent child nodes.

Node scoring function for node selection During program structure search, we prioritize the expansion of the node with the highest potential for program improvement. Since nodes with a higher frequency of errors have greater room for enhancement, we use error count as the scoring function. This metric, which considers both the error rate and the size of the data subset handled by each node, offers a straightforward yet empirically effective approach. Section 5.6 provides empirical evidence demonstrating the efficacy and robustness of this metric against alternatives.

Complete Algorithm The above outline the learning process of a single program (visualized in Figure 5.2. To enhance the full search pipeline, we integrate beam search [PIL23] to avoid getting trapped in local minima. Specifically, each iteration of the learning algorithm maintains and expands B trees, where B represents the beam size.

5.5 Experimental Results

We adopt a comprehensive approach to extensively evaluate the effectiveness of LSPs against various baselines under different settings. Our empirical study is designed to validate the benefits of LSPs over alternative methods by addressing the following research questions:

- **Q1: How does LSP compare against traditional NSPs in expressiveness and interpretability?** We assess this through both quantitative and qualitative evaluations (human studies). (Section 5.5.2)
- **Q2: Does LSP generalize better than traditional NSPs under domain shifts?** This question is explored in detail in (Section 5.5.2).
- **Q3: Is the incorporation of explicit structures beneficial to LSPs?** We compare the structured LSP with vanilla prompt optimization, which exemplifies a special case of LSP with a single LLM module. (Section 5.5.3)
- **Q4: How effective are different LLMs in implementing LSP?** We conduct cross-model experiments to evaluate the performance of various LLMs as the computational backbone for learning and inference in LSP.

5.5.1 General settings

Evaluation For language tasks, we test popular LLMs, including GPT-3.5 (turbo-1104) [Dal21], GPT-4 (1106-preview) [Ope23], and Gemini-M (1.0-pro) [TAB23]. For vision tasks, GPT-4V (1106-vision-preview) and Gemini-Vision (1.5-flash) are utilized. All experiments are repeated with 3 seeds.

Table 5.1: **Classification accuracy comparison with XAI methods on IL-Bench-Vision.** Here, all numbers for LSP are obtained with Gemini-Vision as the learner and inference LLM, except for LSP (GPT-4V) which uses the larger GPT-4V as the learner; Decision Tree, operating directly on pixel data, lacks human interpretability. Key findings include: (1) Our method outperforms XAI baselines with an average accuracy of 95.67%, which is over 10% higher than the nearest competitor. (2) The program generated by LSP also demonstrates superior transferability to human raters, as they are able to reproduce the predictions following rules learned by LSP.

IL-Bench-Vision		Palworld							
MLLM	Method	Mean	Fire-1	Fire-2	Dragon-1	Dragon-2	Electric-1	Electric-2	Water-1
Gemini-M	Decision Tree [CG16]	68.20	91.11 ± 12.57	32.00 ± 9.80	68.33 ± 10.27	48.33 ± 20.95	82.67 ± 6.80	65.33 ± 13.60	66.67 ± 8.50
	ProtoTree [NVS21]	84.33	**100.00 ± 0.00**	62.67 ± 12.36	98.33 ± 2.36	85.00 ± 4.08	**100.00 ± 0.00**	82.67 ± 9.98	61.67 ± 25.93
	LSP	**96.83**	93.33 ± 0.00	**92.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	95.00 ± 5.00	**97.50 ± 2.50**
	LSP (GPT-4V)	95.67	96.67 ± 3.33	90.00 ± 6.00	90.00 ± 10.00	97.50 ± 2.50	**100.00 ± 0.00**	**98.00 ± 2.00**	**97.50 ± 2.50**
Human Rater	ProtoTree [NVS21]	72.74	83.33 ± 16.67	50.0 ± 10.0	**100.0 ± 0.0**	75.0 ± 0.0	83.33 ± 16.67	80.0 ± 0.0	37.5 ± 12.5
	LSP (GPT-4V)	**90.36**	**100.00 ± 0.00**	**70.00 ± 10.00**	**100.00 ± 0.00**	**87.5 ± 12.5**	**100.00 ± 0.00**	**100.00 ± 0.00**	**75.00 ± 25.00**

Implementation details of LSP Our default model of choice is GPT-3.5 for language tasks and Gemini-Vision for vision tasks for cost efficiency, but also examine cross-(M)LLM performance in Appendix. All LLM modules are initialized with an empty instruction "none".

5.5.2 Comparison with traditional interpretable learning methods

We compare LSP with two established models - ProtoTree [NVS21] and Decision Tree [CG16] - both organize prediction process in tree-structured formats. Among existing NSP methods, the closest to ours is ProtoTree - a highly interpretable NSP that learns a discrete binary tree end-to-end, where each node stores an image patch ("prototype") and the edges determine whether the prototype exists within the query image. Note that ProtoTree does not rely on an explicit DSL - we could not

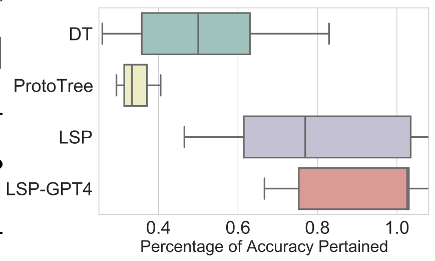


Figure 5.3: **Accuracy retention rate on Out-Of-Distribution variants of IL-Bench-Vision testsets.** We compute the ratio of test accuracy evaluated on OOD datasets to the original test accuracy. LSP shows strong transferability to OOD data. Notably, the version using GPT-4V as the learner retains 90-100% of the original test accuracy.

compare with methods based on explicit DSL since they require domain experts to design those operation, while our goal is to automate the whole process. Since ProtoTree only implements image tasks, this comparison also focus on the vision tasks in IL-Bench.

Expressiveness The expressiveness of the learned programs is evaluated in Table 5.1. LSP (GPT4) outperforms ProtoTree with an average accuracy of 95.67% - over 10% gain. Considering that GPT/Gemini has never observed the images in our datasets before (curated after their knowledge cutoff), this result suggests LSP is capable of formulating effective predictive rules from previously unseen examples.

Interpretability We measure the interpretability of LSPs and NSPs by having human raters make predictions based on visualizations of the learned programs (See Appendix for evaluation protocols). This process essentially "transfers" knowledge from models back to human. Notably, many XAI methods fall short of achieving this level of interpretability, with ProtoTree being a rare exception. As summarized in Table 5.1, the program generated by LSP also demonstrates stronger transferability to human raters, as they are able to largely reproduce the predictions following rules learned by LSP.

Generalization under Domain Shift In contrast to traditional NSP models that rely on parametric memory, LSP utilizes language instructions to encode knowledge. This strategy significantly enhances robustness against variations in visual attributes (domain shifts). To verify this advantage, we examine the transferability of the learned programs to Out-of-Distribution (OOD) data, constructed using GPT-4V (See Appendix for details) As shown in Figure 5.3, LSP

Table 5.2: **Classification accuracy comparison with Prompt Optimization methods on IL-Bench-Language.** Key findings include: (1) LSP achieves $\sim 6\%$ accuracy gain over the second best method, PromptAgent, with comparable search and inference costs. (2) Across synthetic Decision Tree datasets categorized by increasing complexity of oracle decision rules (Easy, Medium, Hard), LSP consistently outperforms other methods in maintaining high accuracy levels, demonstrating its superior ability to reverse-engineer complex rules from observed data.

Text Benchmark	Symbolic						Caption					
Method	Mean Acc	Search Cost	Infer Cost	DT-Easy	DT-Medium	DT-Hard	Waxwing	Waterthrush	Jaeger	Albatross	Blackbird	Swallow
APE [ZMH22]	67.42	270.60s	0.11s	**100.00 ± 0.00**	85.00 ± 4.42	75.67 ± 4.52	50.00 ± 2.72	45.00 ± 3.60	66.11 ± 2.83	48.89 ± 3.14	80.00 ± 3.12	56.11 ± 2.39
OPRO [YWL23a]	55.48	257.86s	0.14s	50.00 ± 1.08	50.17 ± 3.06	30.33 ± 2.62	57.22 ± 2.08	57.22 ± 4.16	76.67 ± 4.71	40.37 ± 3.43	78.06 ± 2.83	55.28 ± 1.04
APO [PIL23]	70.67	270.85s	0.08s	**100.00 ± 0.00**	96.67 ± 4.71	77.83 ± 11.90	56.11 ± 4.78	48.89 ± 4.16	70.00 ± 5.93	54.07 ± 9.70	74.17 ± 2.97	58.33 ± 1.36
TreePrompt[†][SMR23]	65.64	301.52s	0.34s	**100.00 ± 0.00**	83.50 ± 6.68	57.83 ± 5.89	55.00 ± 7.20	53.33 ± 4.91	73.89 ± 1.57	47.78 ± 1.57	65.56 ± 0.39	53.89 ± 2.08
PromptAgent [WLW23]	72.40	220.95s	0.11s	97.67 ± 3.30	88.50 ± 8.44	64.33 ± 20.27	60.56 ± 4.78	56.67 ± 6.24	75.00 ± 3.60	**74.44 ± 6.54**	74.17 ± 1.36	57.22 ± 0.79
LSP (Ours)	**78.53**	232.54	0.13s	99.83 ± 0.24	**99.00 ± 0.82**	**96.83 ± 0.85**	**65.83 ± 4.17**	**62.50 ± 0.83**	**80.00 ± 1.67**	61.11 ± 1.11	**78.75 ± 0.42**	**62.92 ± 0.42**

[†] TreePrompt is a pre-LLM era prompt optimization methods. We adapt this method to support LLMs.

Table 5.3: **Classification accuracy comparison with Prompt Optimization methods on IL-Bench-Vision.** LSP achieves an average accuracy of 96.83%, which is $\sim 20\%$ higher than the 2nd best method (APO).

Vision Benchmark		Palworld						
Method	Mean	Fire-1	Fire-2	Dragon-1	Dragon-2	Electric-1	Electric-2	Water-1
APE [ZMH22]	47.45	60.00 ± 0.00	38.00 ± 18.00	43.33 ± 3.33	42.50 ± 7.50	53.33 ± 0.00	25.00 ± 15.00	70.00 ± 15.00
OPRO [YWL23a]	28.09	13.33 ± 0.00	20.00 ± 0.00	30.00 ± 10.00	25.00 ± 0.00	53.33 ± 20.00	25.00 ± 0.00	30.00 ± 0.00
APO [PIL23]	76.38	70.00 ± 16.67	58.00 ± 10.00	96.67 ± 3.33	77.50 ± 2.50	90.00 ± 10.00	67.50 ± 2.50	75.00 ± 5.00
TreePrompt [SMR23]	67.20	60.00 ± 0.00	50.00 ± 6.00	93.33 ± 6.67	77.50 ± 2.50	53.33 ± 0.00	65.00 ± 20.00	70.00 ± 0.00
PromptAgent [WLW23]	66.33	53.33 ± 40.00	56.00 ± 4.00	96.67 ± 3.33	72.50 ± 17.50	63.33 ± 16.67	55.00 ± 20.00	67.50 ± 27.50
LSP (Ours)	**96.83**	**93.33 ± 0.00**	**92.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**95.00 ± 5.00**	**97.50 ± 2.50**

demonstrates exceptional resilience to domain shifts, compared with ProtoTree.

5.5.3 Comparison with prompt optimization methods

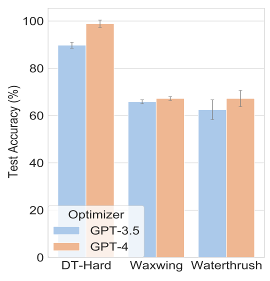
Since there exists a variety of PO method that primarily differ in the search algorithm, we select one most representative method from each major category: Monte Carlo sampling (APE) [ZMH22], evolutionary search (ORPO) [YWL23a], beam search (APO) [PIL23], and tree search (PromptAgent) [WLW23]. We also adapt TreePrompt [SMR23] - a pre-LLM method that fits a classic decision tree to a set of pre-defined prompts - to LLMs. Since the main bottleneck for PO methods is the candidate evaluation, we follow existing works and set the same maximum number of candidate proposals for all methods (100 candidates).

Results The empirical results indicate that incorporating explicit structures significantly enhances performance of the programs on predictive tasks: LSP consistently outperforms all vanilla prompt optimization methods, with a considerable margin of 20.09% and 4.89% over the 2nd best methods on vision and language tasks respectively. The advantages of integrating structured learning are twofold: (1) It simplifies the learning process: LSP benefits from a divide-and-conquer approach where each LLM-module node focuses solely on extracting predictive rules for a specific subset of the data. (2) It streamlines the inference process: We observe that LLMs tend to exhibit hallucination as the complexity of the instructions increases (e.g., multiple conditional clauses. In contrast, LSP mitigates this issue by ensuring that each LLM module contains simpler, more manageable instructions.

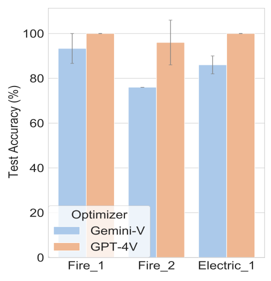
Search cost analysis A key advantage of the structured prediction approach in LSP is that theoretically, it can reduce inference costs when executing oracle decision rules. This efficiency arises because, during prediction, only a small subset of branches is executed for a given test input, and the prompt on each branch is also much simpler due to divide-and-conquer. Consequently, we observe empirically that LSP's search and inference costs are comparable to those of various prompt optimization baselines (Table 5.2).

5.6 Ablation Study

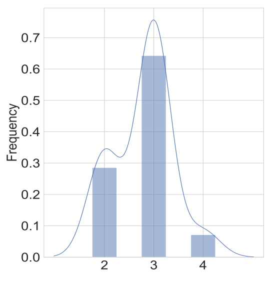
Convergence of LLM-Symbolic Program LSP LSP organizes instructions into a tree-based structure. Such divide-and-conquer strategy simplifies the learning process. To verify this, we also plot the training trajectories for LSP across various tasks. The training trajectory indicates the how fast a model fits the



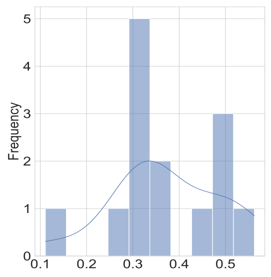
(a) Language Tasks



(b) Vision Tasks



(c) Program Depth



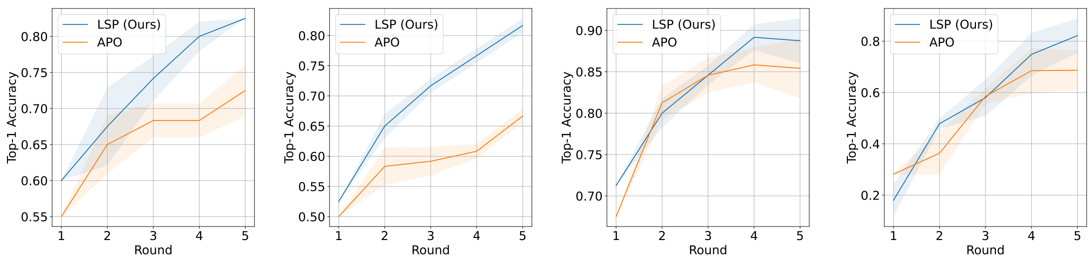
(d) Program Sparsity

Figure 5.4: **(a, b): Stronger LLMs as better LSP learners.** In these experiments, we keep the inference LLM fixed (GPT-3.5 for text and Gemini-V for images) while swapping the learner LLM with GPT-4. With its larger parameter count, GPT-4 consistently achieves better performance in learning LSPs. **(c, d): Statistics of discovered programs.** Averaged from the IL-Bench-Language tasks, the resulting LSPs are generally shallow and sparse, indicating that the final prediction can be reached within only a few steps.

observed examples. As Figure 5.5 demonstrates, LSP not only converges faster but also achieves higher final accuracy compared to models that use unstructured prompting techniques.

Different node scoring functions Table 5.4 summarizes the performance of LSP using three different node scoring functions: (1). Error count. (2). Prediction accuracy. (3). Random scoring. The results suggest that error count performs more consistently across different tasks.

Robustness to meta-prompts LLM's behavior is highly sensitive to prompt formulation, where even minor variations in prompts might lead to significantly different outcomes. To assess the robustness of LSP's performance against variations in the meta-prompt - the prompt used by the learner LLM to generate rules - we conducted experiments with three alternative prompts. These prompts were paraphrased versions generated by distinct LLMs. The results, presented in Table 5.4, indicate that LSP's performance remains consistent across all meta-



(a) Waxwing (b) Waterthrush (c) Blackbird (d) DT-Hard

Figure 5.5: **Convergence of different algorithms across time**. We plot the trajectory of training accuracy against the number of optimization rounds. The API model is GPT-3.5. (1). LSP converges substantially faster than vanilla prompting; (2). The search process does not introduce extra variances.

prompt variants, demonstrating robustness to prompt formulation.

Complexity of discovered programs Our analysis of the statistics of learned programs indicates that the complexity of programs developed by LSP is quite manageable: Most programs can reach a final prediction within just three steps, as illustrated in Figure 5.4c, and the tree structures tend to be sparse, as shown in Figure 5.4d. These observations confirm that although theoretical maximum tree expansion could grow exponentially with depth, in practice, LSPs operate effectively without requiring overly complex structures.

Table 5.4: **Comparison of Different Node Scoring Functions** on three tasks from IL-Bench-Language. Despite its simplicity, error count achieves more consistent performance compared to alternative metrics.

Node Scoring	**DT-Hard**	**Waxwing**	**Waterthrush**
Random	70.50 ± 11.01	62.22 ± 4.78	61.67 ± 1.36
Accuracy	80.33 ± 18.27	**66.11 ± 7.86**	54.44 ± 0.70
Error Count (LSP)	**96.83 ± 0.85**	65.83 ± 4.17	**62.50 ± 0.83**
Meta Prompt	**DT-Hard**	**Waxwing**	**Waterthrush**
Paraphrase-1	97.50 ± 2.12	65.00 ± 4.91	**66.11 ± 3.14**
Paraphrase-2	98.50 ± 0.71	61.67 ± 2.36	62.22 ± 3.93
Paraphrase-3	**99.33 ± 0.62**	62.78 ± 2.83	63.89 ± 0.79
Original (LSP)	96.83 ± 0.85	**65.83 ± 4.17**	62.50 ± 0.83

CHAPTER 6

Prompt Optimization for Diffusion Models

6.1 Problem settings

Large-scale text-based generative models exhibit a remarkable ability to generate novel content conditioned on user input prompts [OWJ22, TLI23, RBL22, RDN22, SCS22, HCS22, YXK22, CZB23]. Despite being trained with huge corpora, there still exists a substantial gap between user intention and what the model interprets [ZMH22, FHF22, RBL22, RKH21, LLY23, OWJ22, RDN22]. The misalignment is even more severe in text-to-image generative models, as they often rely on much smaller and less capable text encoders [RKH21, CBW23, CHL22] than large language models (LLMs). As a result, instructing a large model to produce intended content often requires laborious human efforts in crafting the prompt through trials and errors (a.k.a. Prompt Engineering) [Artar, WMM22, WA22, LC22, ZMH22, HCD22]. To automate this process for language generation, several recent attempts have shown tremendous potential in utilizing LLMs to enhance prompts [PIL23, ZMH22, CCG23, GWG23b, YWL23b, HCD22]. However, efforts on text-to-image generative models remain scarce and preliminary, probably due to the challenges faced by these models' relatively small text encoders in understanding subtle language cues.

DPO-Diff. This paper presents a systematic study of prompt optimization for text-to-image diffusion models. We introduce a novel optimization framework based on the following key observations. *1) Prompt engineering can be formulated as a Discrete Prompt Optimization (DPO) problem over the space of natural languages.* Moreover, the framework can be used to find prompts that either improve (prompt enhancement) or destroy (adversarial attack) the generation process, by simply reversing the sign of the objective function. *2) We show that for diffusion models with classifier-free guidance [HS22], improving the image generation process is more effective when optimizing "negative prompts" [And23, Woo22] than positive prompts.* Beyond the problem formulation of DPO-Diff, where "Diff" highlights our focus on text-to-image diffusion models, the main technical contributions of this paper lie in efficient methods for solving this optimization problem, including the design of compact domain spaces and a gradient-based algorithm.

Compact domain spaces. DPO-Diff's domain space is a discrete search space at the word level to represent prompts. While this space is generic enough to cover any sentence, it is excessively large due to the dominance of words irrelevant to the user input. To alleviate this issue, we design a family of dynamically generated compact search spaces based on relevant word substitutions, for both positive and negative prompts. These subspaces enable efficient search for both prompt enhancement and adversarial attack tasks.

Shortcut gradients for DPO-Diff. Solving DPO-Diff with a gradient-based algorithm requires computing the text gradient, i.e., backpropagating from the generated image, through all inference steps of a diffusion model, and finally to the discrete text. Two challenges arise in obtaining this gradient: 1) This process incurs compound memory-runtime complexity over the number of backward passes through the denoising step, making it prohibitive to run on large-scale diffusion models (e.g., a 870M-parameter Stable Diffusion v1 requires $\sim$750G memory to run backpropagation through 50 inference steps [RBL22]). 2) The embedding lookup tables in text encoders are non-differentiable. To reduce the computational cost in 1), we provide the first generic replacement for the text gradient that bypasses the need to unroll the inference steps in a backward pass, allowing it to be computed with constant memory and runtime. To backpropagate through the discrete embedding lookup table, we continuously relax the categorical word choices to a learnable smooth distribution over the vocabulary, using the Gumbel Softmax trick [GSJ21, JGP16, DY19]. The gradient obtained by this method, termed **Shortcut Gradient**, enables us to efficiently solve DPO-Diff regardless of the number of inference steps of a diffusion model.

To evaluate our prompt optimization method for the diffusion model, we collect and filter a set of challenging prompts from diverse sources including DiffusionDB [WMM22], COCO [LMB14], and ChatGPT [OWJ22]. Empirical results suggest that DPO-Diff can effectively discover prompts that improve (or destroy for adversarial attack) the faithfulness of text-to-image diffusion models, surpassing human-engineered prompts and prior baselines by a large margin. We summarize our primary contributions as follows:

- **DPO-Diff:** A generic framework for prompt optimization as a discrete optimization problem over the space of natural languages, of arbitrary metrics.
- **Compact domain spaces:** A family of dynamic compact search spaces, over which a gradient-based algorithm enables efficient solution finding for the prompt optimization problem.
- **Shortcut gradients:** The first novel computation method to enable backpropagation through the diffusion models' lengthy sampling steps with constant

memory-runtime complexity, enabling gradient-based search algorithms.

- **Negative prompt optimization:** The first empirical result demonstrating the effectiveness of optimizing negative prompts for diffusion models.

6.2 Related work

Text-to-image diffusion models. Diffusion models trained on a large corpus of image-text datasets significantly advanced the state of text-guided image generation [RBL22, RDN22, SCS22, CZB23, YXK22]. Despite the success, these models can sometimes generate images with poor quality. While some preliminary observations suggest that negative prompts can be used to improve image quality [And23, Woo22], there exists no principled way to find negative prompts. Moreover, several studies have shown that large-scale text-to-image diffusion models face significant challenges in understanding language cues in user input during image generation; Particularly, diffusion models often generate images with missing objects and incorrectly bounded attribute-object pairs, resulting in poor "faithfulness" or "relevance" [HCD22, FHF22, LLY23, LLD22]. Existing solutions to this problem include compositional generation [LLD22], augmenting diffusion model with large language models [YWL23b], and manipulating attention masks [FHF22]. As a method orthogonal to them, our work reveals that negative prompt optimization can also alleviate this issue.

Prompt optimization for text-based generative models. Aligning a pretrained large language model (LLM) with human intentions is a crucial step toward unlocking the potential of large-scale text-based generative models [OWJ22, RBL22]. An effective line of training-free alignment methods is prompt optimization (PO) [ZMH22]. PO originated from in-context learning [Dal21], which is mainly concerned with various arrangements of task demonstrations. It later evolves into automatic prompt engineering, where powerful language models are utilized to refine prompts for certain tasks [ZMH22, PIL23, YWL23b, PIL23, HCD22]. While PO has been widely explored for LLMs, efforts on diffusion models remain scarce. The most relevant prior work to ours is Promptist [HCD22], which finetunes an LLM via reinforcement learning from human feedback [OWJ22] to augment user prompts with artistic modifiers (e.g., high-resolution, 4K) [Artar], resulting in aesthetically pleasing images. However, the lack of paired contextual-aware data significantly limits its ability to follow the user intention (Figure 6.2b).

Backpropagating through the sampling steps of diffusion models. Text-to-image diffusion models generate images via a progressive denoising process, making multiple passes through the same network [HJA20]. When a loss is applied to the output image, computing the gradient w.r.t. any model component (text, weight, sampler, etc.) requires backpropagating through all the sampling steps. This process incurs compound complexity over the number of backward passes in both memory and runtime, making it infeasible to run on regular commercial devices. Existing efforts achieve constant memory via gradient checkpointing [WCH21] or solving an augmented SDE problem [NGH22], at the expense of even higher runtime. In this paper, we propose a novel solution to compute a "shortcut" gradient, resulting in constant complexity in both memory and runtime.

6.3 Preliminaries on diffusion model

We provide a brief overview of relevant concepts in diffusion models, and refer the reader to [Luo22] for detailed derivations.

Denoising diffusion probablistic models. On a high level, diffusion models [HJA20] are a type of hierarchical variational autoencoder [SRM16] that generates samples by reversing a progressive noising process. Let $\boldsymbol{x}_0 \cdots \boldsymbol{x}_T$ be a series of intermediate samples at increasing noise levels, the noising (forward) process can be expressed as the following Markov chain:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}) \quad t = 1 \sim T, \tag{6.1}$$

where β is a scheduling variable. Using Gaussian reparameterization, sampling $\boldsymbol{x}_t$ from $\boldsymbol{x}_0$ can be completed in a single step:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \alpha_t = 1 - \beta_t \text{ and } \bar{\alpha}_t = \prod\nolimits_{i=1}^{t} \alpha_i, \tag{6.2}$$

where ϵ is a standard Gaussian error. The reverse process starts with a standard Gaussian noise, $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and progressively denoises it using the following joint distribution:

$$p_\theta(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_T)\prod\nolimits_{t=1}^{T} p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \;\text{ where }\; p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_\theta(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}).$$

While the mean function $\mu_\theta(\boldsymbol{x}_t, t)$ can be parameterized by a neural network (e.g., UNet [RBL22, RFB15]) directly, prior studies found that modeling the residual error $\epsilon(\boldsymbol{x}_t, t)$ instead works better empirically [HJA20]. The two strategies are mathematically equivalent as $\mu_\theta(\boldsymbol{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon(\boldsymbol{x}_t, t))$.

Classifier-free guidance for conditional generation. The above formulation can be easily extended to conditional generation via classifier-free guidance [HS22], widely adopted in contemporary diffusion models. At each sampling step, the predicted error $\tilde{\epsilon}$ is obtained by subtracting the unconditional error from the conditional error (up to a scaling factor w):

$$\tilde{\epsilon}_\theta(\boldsymbol{x}_t, c(s), t) = (1 + w)\epsilon_\theta(\boldsymbol{x}_t, c(s), t) - w\epsilon_\theta(x_t, c(\text{``''}), t), \tag{6.3}$$

where $c(s)$ is the conditional signal of text s, and the unconditional prior $c(\text{``''})$ is obtained by passing an empty string to the text encoder. If we replace this empty string with an actual text, then it becomes a "negative prompt" [And23, Woo22], indicating what to exclude from the generated image.

6.4 DPO-Diff: Discrete Prompt Optimization for diffusion models

This section lays out the components of DPO-Diff framework. Section 4.1 explains how to formulate the problem into optimization over the text space. This is followed by the full algorithm for solving this optimization, including the compact search space in Section 4.2, and the gradient-based search algorithm in Section 4.3.

6.4.1 Framework Overview

Our main insight is that prompt engineering can be formulated as a discrete optimization problem in the language space, called DPO-Diff. Concretely, we represent the problem domain $\mathcal{S}$ as a sequence of M words w_i from a predefined vocabulary $\mathcal{V}$: $\mathcal{S} = \{w_1, w_2, \dots w_M | \forall i,\ w_i \in \mathcal{V}\}$. This space is generic enough to cover all possible sentences of lengths less than M (when the empty string is present). Let $G(s)$ denote a text-to-image generative model, and s_{user}, s denote the user input and optimized prompt, respectively. The optimization problem can be written as

$$\min_{s \in \mathcal{S}} \mathcal{L}(G(s), s_{user}) \qquad \text{s.t.} \quad d(s, s_{user}) \leq \lambda, \tag{6.4}$$

where $\mathcal{L}$ can be any objective function that measures the effectiveness of the learned prompt when used to generate images, and $d(\cdot, \cdot)$ is an optional constraint function that restricts the distance between the optimized prompt and the user input. Following previous works [HCD22], we use clip loss CLIP(I, s_{user}) [o22] to measure the alignment between the generated image I and the user prompt s_{user} — the **Faithfulness** of the generation process. Like any automatic evaluator for generative models, the clip score is certainly not free from errors. However, through the lens of human evaluation, we find that it is mostly aligned with human judgment for our task.

This DPO-Diff framework is versatile for handling both prompt improvement and adversarial attack. Finding adversarial prompts can help diagnose the failure modes of generative models, as well as augment the training set to improve a model's robustness via adversarial training [MMS17b]. We define adversarial prompts for text-to-image generative models as follows.

Definition 6.4.1. *Given a user input s_{user}, an adversarial prompt s_{adv} is a text input that is semantically similar to s_{user}, yet causes the model to generate images that cannot be described by s_{user}.*

Intuitively, Definition refadv aims at perturbing the user prompt without changing its overall meaning to destroy the prompt-following ability of image generation. Formally, the adversarial prompt is a solution to the following problem,

$$\min_{s \in \mathcal{S}} \; -\mathcal{L}(G(s), s_{user}) \quad \text{s.t. } d(s, s_{user}) \leq \lambda \tag{6.5}$$

where the first constraint enforces semantic similarity.

To apply DPO-Diff to adversarial attack, we can simply add a negative sign to $\mathcal{L}$, and restrict the distance between s and s_{user} through d. This allows equation 6.5 to produce an s that increases the distance between the image and the user prompt, while still being semantically similar to s_{user}.

6.4.2 Compact search space design for efficient prompt discovery

While the entire language space facilitates maximal generality, it is also unnecessarily inefficient as it is popularized with words irrelevant to the task. We propose a family of compact search spaces that dynamically extracts a subset of task-relevant words to the user input.

6.4.2.1 Application 1: Discovering adversarial prompts for model diagnosis

Synonym Space. In light of the first constraint on semantic similarity in equation 6.5, we build a search space for the adversarial prompts by substituting each word in the user input s_{user} with its synonyms [ASE18], preserving the meaning of the original sentence. The synonyms can be found by either dictionary lookup or querying ChatGPT.

6.4.2.2 Application 2: Discovering enhanced prompts for image generation

While the Synonym Space is suitable for attacking diffusion models, we found that it performs poorly on finding improved prompts. This is in contradiction to LLMs where rephrasing user prompts can often lead to substantial gains [ZMH22]. One plausible reason is that contemporary diffusion models often rely on a small-scale clip-based text encoders [RKH21, CBW23, CHL22], which are weaker than LLMs with many known limitations in understanding subtle language cues [FHF22, LLD22, YWL23b].

Inspired by these observations, we propose a novel solution to optimize for **negative** prompts instead — a unique concept that rises from classifier-free guidance [HS22] used in diffusion models (Section 6.3). To the best of our knowledge, we provide the first exploratory work on automated negative prompt optimization.

Antonym Space. We propose to build the space of negative prompts based on the antonyms of each word, as opposed to the Synonym Space for adversarially attacking the model. Intuitively, the model's output image can safely exclude the content with the opposite meaning to the words in the user input, so it instead amplifies the concepts presented in the positive prompt. Similar to synonyms, the antonyms of words in a user prompt can be obtained via dictionary lookup or ChatGPT.

Negative Prompt Library (NPLib). We further crawl and filter a library of human-crafted generic negative prompts to augment the antonym space. This augmentation enhances the image generation quality and provides a safeguard

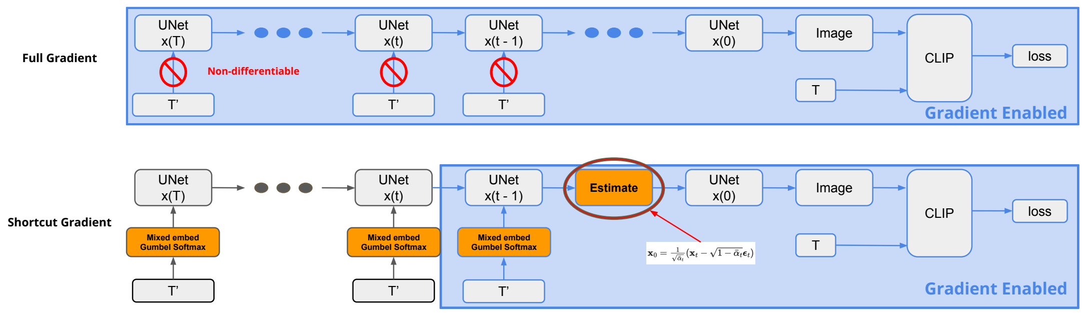


Figure 6.1: Computational procedure of Shortcut Gradient (Bottom) v.s. Full Gradient (Top) on text.

when a user input has a small number of high-quality antonyms. We term our library **NPLib**, which will be released with our codebase.

6.4.3 A Gradient-based algorithm as a DPO-Diff solver

Due to the query efficiency of white-box algorithms leveraging gradient information, we also explore a gradient-based method to solve equation 6.4 and equation 6.5. However, obtaining this text gradient is non-trivial due to two major challenges. 1) Backpropagating through the sampling steps of the diffusion inference process incurs high complexity w.r.t. memory and runtime, making it prohibitively expensive to obtain gradients. [1] 2) The embedding lookup table used in the text encoder is non-differentiable. Section 6.4.3.1 introduces the Shortcut Gradient, a replacement for text gradient with constant memory and runtime. Section 6.4.3.2 discusses how to backpropagate through the embedding lookup table via continuous relaxation. Section 6.4.3.3 describes how to sample from the learned distribution via evolutionary search.

[1]The text gradient is completely different from "Textual Inversion" [GAA22], as the later can be obtained in the same way as regular gradients used in diffusion training, requiring only a single backward pass.

6.4.3.1 Backpropagating through diffusion sampling steps

Shortcut gradient. Backpropagating through the diffusion model inference process requires executing multiple backward passes through the generator network [WCH21, NGH22]; For samplers with 50 inference steps (e.g., DDIM [SME20]), it raises the runtime and memory cost by **50 times** compared to a single diffusion training step. To alleviate this issue, we propose Shortcut Gradient, an efficient replacement for text gradients that can be obtained with constant memory and runtime.

The key idea behind the Shortcut Gradient is to reduce gradient computation from all to K sampling steps, resulting in a constant number of backward passes. The entire pipeline (Figure 6.1) can be divided into three steps:

(1) Sampling without gradient from step T (noise) to t. In the first step, we simply disable gradients up to step t. No backward pass is required for this step.

(2) Enable gradient from t to $t-K$. We enable the computational graph for a backward pass for K step starting at t.

(3) Estimating $\boldsymbol{x}_0$ from $\boldsymbol{x}_{t-K}$ through closed-form solution. To bypass the gradient computation in the remaining steps, simply disabling the gradient like (1) is no longer valid because otherwise the loss applied to $\boldsymbol{x}_0$ could not propagate back. Directly decoding and feeding the noisy intermediate image $\boldsymbol{x}_{t-K}$ to the loss function is also not optimal due to distribution shift [DN21]. Instead, we propose to use the current estimate of $\boldsymbol{x}_0$ from $\boldsymbol{x}_{t-K}$ to bridge the gap. From the forward equation of the diffusion model, we can derive a connection between the final image $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{x}_{t-K}$ as $\hat{\boldsymbol{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_{t-K}}}(\boldsymbol{x}_{t-K} - \sqrt{1-\bar{\alpha}_{t-K}}\boldsymbol{\epsilon}_{t-K})$. In this way, the Jacobian of $\hat{\boldsymbol{x}}_0$ w.r.t. $\boldsymbol{x}_{t-K}$ can be computed analytically, with complexity independent of t.

- **Remark 1**: The estimation is not a trick — it directly comes from a mathematically equivalent interpretation of the diffusion model, where each inference step can be viewed as computing $\hat{\boldsymbol{x}}_0$ and plugging it into $q(\boldsymbol{x}_{t-K}|\boldsymbol{x}_t, \hat{\boldsymbol{x}}_0)$ to obtain the transitional probability.

- **Remark 2**: The computational cost of the Shortcut gradient is controlled by K. Moreover, when we set $t = T$ and $K = T$, it becomes the full-text gradient.

Strategy for selecting t. At each iteration, we select a t and compute the gradient of the loss over text embeddings using the above mechanism. Empirically, we found that setting it around the middle point and progressively reducing it produces the most salient gradient signals.

6.4.3.2 Backpropagating through embeddings lookup table

In diffusion models, a tokenizer transforms text input into indices, which will be used to query a lookup table for corresponding word embeddings. To allow further propagating gradients through this non-differentiable indexing operation, we relax the categorical choice of words into a continuous probability of words. It can be viewed as learning a "distribution" over words. We parameterize the distribution using Gumbel Softmax [JGP16] with uniform temperature ($\eta = 1$):

$$\tilde{e} = \sum_{i=1}^{|\mathcal{V}|} p(w = i; \alpha) e_i \ , \quad p(w = i; \alpha) = \frac{\exp\left((\log \alpha_i + g_i)/\eta\right)}{\sum_{i=1}^{|\mathcal{V}|} \exp\left((\log \alpha_i + g_i)/\eta\right)}, \tag{6.6}$$

where α (a $|\mathcal{V}|$-dimensional vector) denotes the learnable parameter, g denotes the Gumbel random variable, e_i is the embedding of word i, and $\tilde{e}$ is the mixed embedding.

6.4.3.3 Efficient sampling with Evolutionary Search

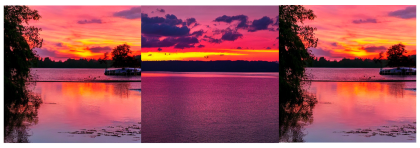
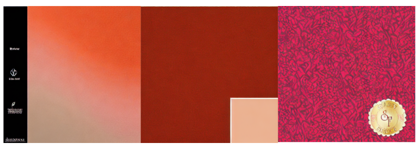
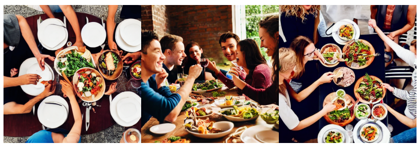
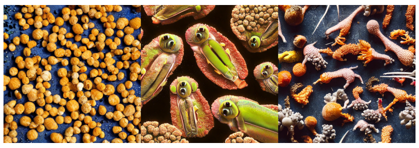
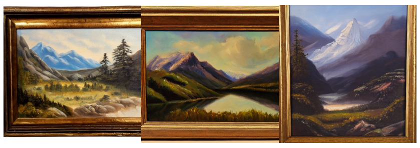
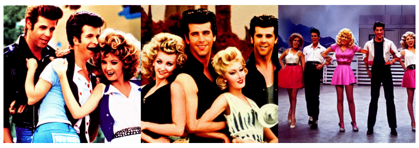
After learning a distribution over words, we can further sample candidate prompts from it via random search. However, random search is sample inefficient, as evidenced in AutoML literatures [WDZ19]. We thus adopt an evolutionary algorithm [Gol89] to search for the best candidate instead, which is simple to implement yet demonstrates strong performance. We view sentences as DNAs and the word choices as nucleotides; To initiate the evolution process, we fill the population with samples from the learned distribution and apply a traditional Evolution Algorithm to find the best one.

Remark: Blackbox Optimization. When the internal state of the model is accessible (e.g., the model owner provides prompt suggestions), gradient information can greatly speed up the search process. In cases where only forward API is available, our Evolutionary Search (ES) module can be used as a stand-alone black-box optimizer, thereby extending the applicability of our framework. We ablate this choice in Section 6.6.1, where ES archives descent results given enough queries.

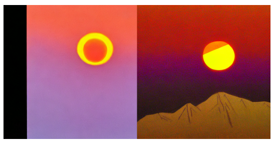
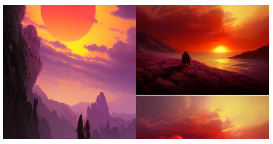
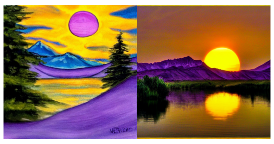
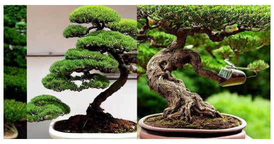
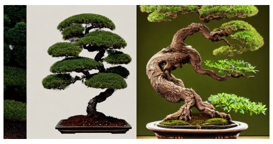
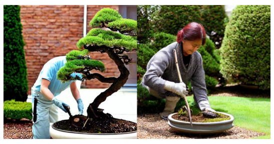
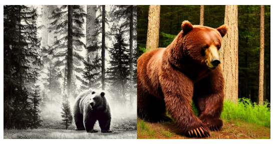
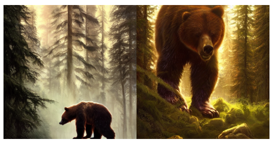
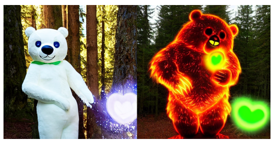
6.5 Experiments

6.5.1 Experimental Setup

Dataset preparation. To encourage semantic diversity, we collect a prompt dataset from three sources: DiffusionDB [WMM22], ChatGPT generated prompts [OWJ22], and COCO [LMB14]. For each source, we filter 100 **"hard prompts"** with a clip loss higher (lower for adversarial attack) than a threshold, amounting to **600 prompts** in total for two tasks.

User Input	DPO-Diff
A vibrant sunset casting hues of orange and pink. 	The vibrant sundown casting tones of orange plus blush. 
A group of friends gather around a table for a meal. 	A party of friends cluster around a surface for a food 
oil painting of a mountain landscape 	grease picture illustrating one mountain view 

(a) Adversarial Attack

User Input	Promptist - Modifiers	Negative Prompts by DPO-Diff
The yellow sun was descending beyond the violet peaks, coloring the sky with hot shades. 	by Greg Rutkowski and Raymond Swanland, ..., ultra realistic digital art 	red, soaring, red, valleys, white, floor, Plain, body, focus, surreal 
A dedicated gardener tending to a meticulously manicured bonsai tree. 	intricate, elegant, highly detailed, ..., sharp focus, illustration, by justin gerard and artgerm, 8 k 	irresponsible, overlooking, randomly, huge, herb, Cropped, complex, faces, photoshopped 
magical shapeshifting large bear with glowing magical marks and wisps of magic, forest... 	D&D, fantasy, cinematic lighting, ..., art by artgerm and greg rutkowski and alphonse mucha 	normal, stable, tiny, elephant, ..., heaps, tundra, advance, Boring, black, expired, perspective 

(b) Prompt Improvement

Figure 6.2: Images generated by user input and improved negative prompts using Stable Diffusion.

Table 6.1: Quantitative evaluation of DPO discovered prompts. For each method, we report the average spherical clip loss of the generated image and user input over all prompts. Note that spherical clip loss normally ranges from 0.75 - 0.85, hence a change above 0.05 is already substantial.

Prompt	**DiffusionDB**	**COCO**	**ChatGPT**
User Input	0.76 ± 0.03	0.77 ± 0.03	0.77 ± 0.02
DPO-Adv	$\mathbf{0.86 \pm 0.05}$	$\mathbf{0.94 \pm 0.04}$	$\mathbf{0.95 \pm 0.05}$

(a) Adversarial Attack $\uparrow$

Prompt	**DiffusionDB**	**COCO**	**ChatGPT**
User Input	0.87 ± 0.02	0.87 ± 0.01	0.84 ± 0.01
Manual	0.89 ± 0.04	0.88 ± 0.02	0.86 ± 0.03
Promptist	0.88 ± 0.02	0.87 ± 0.03	0.85 ± 0.02
DPO	$\mathbf{0.81 \pm 0.03}$	$\mathbf{0.82 \pm 0.02}$	$\mathbf{0.78 \pm 0.03}$

(b) Prompt Improvement $\downarrow$

Evaluation. All methods are evaluated quantitatively using the clip loss [CBK22], complemented by qualitative evaluation by human judgers. We select Stable Diffusion v1-4 as the base model. Each prompt is evaluated under three random seeds (shared across different methods).

Optimization Parameters. We use the spherical clip loss [o22] as the objective function, which ranges between 0.75 and 0.85 for most inputs. The K for the shortcut gradient is set to 1 since we found that it already produces effective supervision signals. To generate the search spaces, we prompt ChatGPT for at most 5 substitutes of each word in the user prompt. Furthermore, we use a fixed set of hyperparameters for both prompt improvement and adversarial attacks.

6.5.2 Discovering adversarial prompts

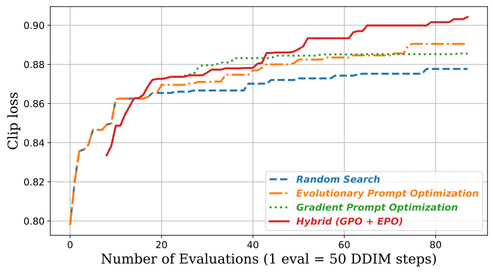
Unlike RLHF-based methods for enhancing prompts (e.g., Promptist [HCD22]) that requires fine-tuning a prompt generator when adapting to a new task, DPO-Diff can be seamlessly applied to finding adversarial prompts by simply reversing the sign of the objective function. These adversarial prompts can be used to diagnose the failure modes of diffusion models or improve their robustness via adversarial training [MMS17b].

Table 6.1a shows adversarial prompt results. Our method is able to perturb the original prompt to adversarial directions, resulting in a substantial increase in the clip loss. Figure 6.2a visualizes a set of intriguing images generated by the adversarial prompts. We can see that DPO-Diff can effectively explore the text regions where Stable Diffusion fails to interpret.

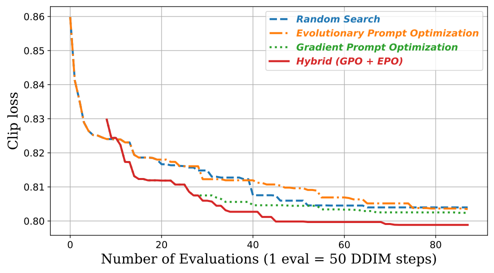
6.5.3 Prompt optimization for improving Stable Diffusion

In this section, we apply DPO-Diff to discover refined prompts to improve the relevance of generated images with user intention. We compare our method with three baselines: (1) User Input. (2) Human Engineered Prompts (available only on DiffusionDB) [WMM22]. (3) Promptist [HCD22], trained to mimic the human-engineered prompt provided in DiffusionDB. For DiffusionDB, following Promptist [HCD22], we extract user input by asking ChatGPT to remove all trailing aesthetics from the original human-engineered prompts.

Table 6.1b summarizes the result. We found that both human-engineered and Promptist-optimized prompts do not improve the relevance. The reason is that they change the user input by merely adding a set of aesthetic modifiers to the original prompt, which are irrelevant to the semantics of user input and cannot



(a) Attack



(b) Improve

Figure 6.3: Learning curves of different search algorithms in solving DPO.

improve the generated images' faithfulness to user intentions. This can be further evidenced by the examples in Figure 6.2b.

Human Evaluation. We further asks 5 human judgers to evaluate the generated images of each method on a manually filtered subset of 100 prompts. When evaluated based on how well the generated image can be described by the user input, the prompts discovered by DPO-Diff achieved a 64% win rate, 15% draw, and 21% loss rate compared with Promptist.

6.6 Ablation Study

We conduct ablation studies on DPO-Diff using 30 randomly selected prompts (10 from each source). Each algorithm is run with 4 seeds to account for the randomness in the search phase.

6.6.1 Comparison of different search algorithms.

We compare four search algorithms for DPO-Diff: Random Search (RS), Evolution Prompt Optimization (EPO), Gradient-based Prompt Optimization (GPO), and the full algorithm (GPO + ES). Figure 6.3 shows their performance under

Table 6.2: Quantitative evaluation of optimizing negative prompts (w/ Antonyms Space) and positive prompts (w/ Synonym Space) for Stable Diffusion.

Prompt	**DiffusionDB**	**ChatGPT**	**COCO**
User Input	0.8741 ± 0.0203	0.8159 ± 0.0100	0.8606 ± 0.0096
Positive Prompt	0.8747 ± 0.0189	0.8304 ± 0.0284	0.8624 ± 0.0141
Negative Prompt	**0.8579 ± 0.0242**	**0.8133 ± 0.0197**	**0.8403 ± 0.0210**

different search budgets (number of evaluations)[2]; While GPO tops EPO under low budgets, it also plateaus quicker as randomly drawing from the learned distribution is sample-inefficient. Combining GPO with EPO achieves the best overall performance.

6.6.2 Negative prompt v.s. positive prompt optimization

One finding in our work is that optimizing negative prompts (Antonyms Space) is more effective than positive prompts (Synonyms Space) for Stable Diffusion. To verify the strength of these spaces, we randomly sample 100 prompts for each space and compute their average clip loss of generated images. Table 6.2 suggests that Antonyms Space contains candidates with consistently lower clip loss than Synonyms Space.

[2]Since the runtime of backpropagation through one-step diffusion sampling is negligible w.r.t. the full sampling process (50 steps for DDIM sampler), we count it the same as one inference step.

CHAPTER 7

Conclusion

The journey of intelligence, from simple neural systems to the creation of advanced artificial intelligence, underscores the potential of self-evolving mechanisms. AutoML embodies this principle by automating the search for optimal machine learning components. While initially conceived as a tool for pipeline optimization, its methodologies are now being repurposed and expanded in the context of large foundation models.

This thesis establishes that AutoML's core principles align with the challenges and opportunities presented by large-scale generative models. By framing tasks like novel architecture, optimizer, and prompt engineering as instances of a broader search problem, we demonstrate that AutoML can adapt to leverage the latent capabilities of large language models and diffusion models. The exploration of efficient search algorithms, generic search spaces, and novel optimization methodologies reveals a promising avenue for maximizing the utility of foundation models.

The findings presented here illuminate the role of AutoML in shaping the future of machine learning, bridging foundational advancements with real-world applications. As we move toward more scalable and adaptive AI systems, the convergence of AutoML and foundational models offers a pathway toward recreating self-evolving mechanisms, taking us one step closer to realizing the vision of AGI.

Bibliography

[ACO17] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. "Towards better understanding of gradient-based attribution methods for deep neural networks." *arXiv preprint arXiv:1711.06104*, 2017.

[ADG16] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. "Learning to learn by gradient descent by gradient descent." *Advances in neural information processing systems*, **29**, 2016.

[AFL23] Griffin Adams, Alexander Fabbri, Faisal Ladhak, Eric Lehman, and Noémie Elhadad. "From sparse to dense: GPT-4 summarization with chain of density prompting." *arXiv preprint arXiv:2309.04269*, 2023.

[Aka17] Charles Sutton Akash Srivastava. "Autoencoding Variational Inference For Topic Models." In *International Conference on Learning Representations*, 2017.

[AMD21] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. "Zero-cost proxies for lightweight NAS." In *ICLR*, 2021.

[And23] Andrew. "How to use negative prompts?", 2023.

[Artar] Lexica Art. "Lexica.", Year.

[ASE18] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. "Generating natural language adversarial examples." *arXiv preprint arXiv:1804.07998*, 2018.

[BHX19] Kaifeng Bi, Changping Hu, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. "Stabilizing DARTS with Amended Gradient Estimation on Architectural Parameters.", 2019.

[Bis16] Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer New York, 2016.

[BKZ18] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. "Understanding and Simplifying One-Shot Architecture Search." In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 550–559, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[BLR18] Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. "SMASH: One-Shot Model Architecture Search through HyperNetworks." In *International Conference on Learning Representations*, 2018.

[BZV17] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. "Neural optimizer search with reinforcement learning." In *International Conference on Machine Learning*, pp. 459–468. PMLR, 2017.

[CAS20] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. "RobustBench: a standardized adversarial robustness benchmark." *arXiv preprint arXiv:2010.09670*, 2020.

[CBK22] Katherine Crowson, Stella Biderman, Daniel Kornis, Dashiell Stander, Eric Hallahan, Louis Castricato, and Edward Raff. "Vqgan-clip: Open domain image generation and editing with natural language guidance." In *European Conference on Computer Vision*, pp. 88–105. Springer, 2022.

[CBW23] Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. "Reproducible scaling laws for contrastive language-image learning." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2818–2829, 2023.

[CCC21] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. "Learning to optimize: A primer and a benchmark." *arXiv preprint arXiv:2103.12828*, 2021.

[CCG23] Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. "InstructZero: Efficient Instruction Optimization for Black-Box Large Language Models." *arXiv preprint arXiv:2306.03082*, 2023.

[CG16] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[CGF19] Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. "Probabilistic Neural Architecture Search." *arXiv: 1902.05116*, 2019.

[CGW21] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. "Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective." In *International Conference on Learning Representations*, 2021.

[CH20a] Xiangning Chen and Cho-Jui Hsieh. "Stabilizing Differentiable Architecture Search via Perturbation-based Regularization." In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1554–1565. PMLR, 13–18 Jul 2020.

[CH20b] Francesco Croce and Matthias Hein. "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks." In *ICML*, 2020.

[CHC17] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. "Learning to learn without gradient descent by gradient descent." In *International Conference on Machine Learning*, pp. 748–756. PMLR, 2017.

[CHL22] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. "Scaling instruction-finetuned language models." *arXiv preprint arXiv:2210.11416*, 2022.

[CLH22] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Yao Liu, Kaiyuan Wang, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. "Evolved Optimizer for Visio." In *AutoML 2022 Workshop*, 2022.

[CLQ20] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. "Adabert: Task-adaptive bert compression with differentiable neural architecture search." *arXiv preprint arXiv:2001.04246*, 2020.

[CLS19] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks." In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

[CRS19] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. "Unlabeled data improves adversarial robustness." *Advances in Neural Information Processing Systems*, **32**, 2019.

[CWC21] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. "DrNAS: Dirichlet neural architecture search." In *ICLR*, 2021.

[CXW19] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1294–1303, 2019.

[CZ21] Guofeng Cui and He Zhu. "Differentiable synthesis of program architectures." *Advances in Neural Information Processing Systems*, **34**:11123–11135, 2021.

[CZB23] Huiwen Chang, Han Zhang, Jarred Barber, AJ Maschinot, Jose Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Murphy, William T Freeman, Michael Rubinstein, et al. "Muse: Text-to-image generation via masked generative transformers." *arXiv preprint arXiv:2301.00704*, 2023.

[CZH19] Han Cai, Ligeng Zhu, and Song Han. "ProxylessNAS: Direct Neural

Architecture Search on Target Task and Hardware." In *International Conference on Learning Representations*, 2019.

[Dal21] Robert Dale. "GPT-3: What's it good for?" *Natural Language Engineering*, **27**(1):113–118, 2021.

[Dav03] Michael I. Jordan David M. Blei, Andrew Y. Ng. "Latent Dirichlet Allocation." *The Journal of Machine Learning Research*, Mar 2003.

[DCA20] Lukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. "BRP-NAS: Prediction-based NAS using GCNs." In *NeurIPS*, 2020.

[DG17] Piotr Dabkowski and Yarin Gal. "Real time image saliency for black box classifiers." In *Advances in Neural Information Processing Systems*, pp. 6967–6976. NeurIPS, 2017.

[DN21] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis." *Advances in neural information processing systems*, **34**:8780–8794, 2021.

[DWH22] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. "Rlprompt: Optimizing discrete text prompts with reinforcement learning." *arXiv preprint arXiv:2205.12548*, 2022.

[DY19] Xuanyi Dong and Yi Yang. "Searching for a robust neural architecture in four gpu hours." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1761–1770, 2019.

[DY20] Xuanyi Dong and Yi Yang. "NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search." In *International Conference on Learning Representations (ICLR)*, 2020.

[EIS19] Logan Engstrom, Andrew Ilyas, Hadi Salman, Shibani Santurkar, and Dimitris Tsipras. "Robustness (Python Library).", 2019.

[ENP19] Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. "Write, execute, assess: Program synthesis with a repl." *Advances in Neural Information Processing Systems*, **32**, 2019.

[FBM23] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. "Promptbreeder: Self-referential self-improvement via prompt evolution." *arXiv preprint arXiv:2309.16797*, 2023.

[FH17] Nicholas Frosst and Geoffrey Hinton. "Distilling a neural network into a soft decision tree." *arXiv preprint arXiv:1711.09784*, 2017.

[FHF22] Weixi Feng, Xuehai He, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Pradyumna Narayana, Sugato Basu, Xin Eric Wang, and William Yang Wang. "Training-free structured diffusion guidance for compositional text-to-image synthesis." *arXiv preprint arXiv:2212.05032*, 2022.

[GAA22] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. "An image is worth one word: Personalizing text-to-image generation using textual inversion." *arXiv preprint arXiv:2208.01618*, 2022.

[GCS04] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis.* Chapman and Hall/CRC, 2nd ed. edition, 2004.

[GLD22] Tanya Goyal, Junyi Jessy Li, and Greg Durrett. "News summarization and evaluation in the era of gpt-3." *arXiv preprint arXiv:2209.12356*, 2022.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning 1st Edition.* Addison-Wesley Professional, 1989.

[GQU20] Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. "Uncovering the limits of adversarial training against norm-bounded adversarial examples." *arXiv preprint arXiv:2010.03593*, 2020.

[GSJ21] Chuan Guo, Alexandre Sablayrolles, Hervé Jégou, and Douwe Kiela. "Gradient-based adversarial attacks against text transformers." *arXiv preprint arXiv:2104.13733*, 2021.

[GSS17] Klaus Greff, Rupesh K. Srivastava, and Jürgen Schmidhuber. "Highway and Residual Networks learn Unrolled Iterative Estimation." In *International Conference on Learning Representations (ICLR)*, 2017.

[GWG23a] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers." *arXiv preprint arXiv:2309.08532*, 2023.

[GWG23b] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. "Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers." *arXiv preprint arXiv:2309.08532*, 2023.

[GZM19] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. "Single Path One-Shot Neural Architecture Search with Uniform Sampling.", 2019.

[HCD22] Yaru Hao, Zewen Chi, Li Dong, and Furu Wei. "Optimizing prompts for text-to-image generation." *arXiv preprint arXiv:2212.09611*, 2022.

[HCS22] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. "Imagen video: High definition video generation with diffusion models." *arXiv preprint arXiv:2210.02303*, 2022.

[HFZ20] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. "Open graph benchmark: Datasets for machine learning on graphs." *Advances in neural information processing systems*, **33**:22118–22133, 2020.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in neural information processing systems*, **33**:6840–6851, 2020.

[HKK17] Dongyoon Han, Jiwhan Kim, and Junmo Kim. "Deep Pyramidal Residual Networks." In *CVPR*, 2017.

[HLM17] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks." *2017*

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[HS22] Jonathan Ho and Tim Salimans. “Classifier-free diffusion guidance.” *arXiv preprint arXiv:2207.12598*, 2022.

[HSY23] Cho-Jui Hsieh, Si Si, Felix X Yu, and Inderjit S Dhillon. “Automatic engineering of long prompts.” *arXiv preprint arXiv:2311.10117*, 2023.

[HWL21] Shoukang Hu, Ruochen Wang, HONG Lanqing, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. “Generalizing Few-Shot NAS with Gradient Matching.” In *International Conference on Learning Representations*, 2021.

[HXZ20] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. “DSNAS: Direct Neural Architecture Search without Parameter Retraining.” In *CVPR*, 2020.

[HZC17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.” *arXiv: 1704.04861*, 2017.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition.” In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax.” *arXiv preprint arXiv:1611.01144*, 2016.

[JLP19] Weonyoung Joo, Wonsung Lee, Sungrae Park, , and Il-Chul Moon. "Dirichlet Variational Autoencoder.", 2019.

[JT16] Kevin Jamieson and Ameet Talwalkar. "Non-stochastic best arm identification and hyperparameter optimization." In *AISTATS*, 2016.

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[KNZ19] Samuel Kessler, Vu Nguyen, Stefan Zohren, and Stephen Roberts. "Hierarchical Indian Buffet Neural Networks for Bayesian Continual Learning.", 2019.

[KP20] Tin Kramberger and Božidar Potočnik. "LSUN-Stanford car dataset: enhancing large-scale car image datasets using deep learning for usage in GAN training." *Applied Sciences*, **10**(14):4913, 2020.

[KT19] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

[LC19] Guillaume Lample and François Charton. "Deep Learning For Symbolic Mathematics." In *International Conference on Learning Representations*, 2019.

[LC22] Vivian Liu and Lydia B Chilton. "Design guidelines for prompt engineering text-to-image generative models." In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pp. 1–23, 2022.

[LHZ20] Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. "A Neural Dirichlet Process Mixture Model for Task-Free Continual Learning," in International Conference on Learning Representations." In *ICLR*, 2020.

[LKB20] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. "Geometry-Aware Gradient Algorithms for Neural Architecture Search." *arXiv: 2004.07802*, 2020.

[LLD22] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. "Compositional visual generation with composable diffusion models." In *European Conference on Computer Vision*, pp. 423–439. Springer, 2022.

[LLY23] Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. "LLM-grounded Diffusion: Enhancing Prompt Understanding of Text-to-Image Diffusion Models with Large Language Models." *arXiv preprint arXiv:2305.13655*, 2023.

[LM16] Ke Li and Jitendra Malik. "Learning to optimize." *arXiv preprint arXiv:1606.01885*, 2016.

[LMB14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context." In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

[LQD20] Guohao Li, Guocheng Qian, Itzel C. Delgadillo, Matthias Muller, Ali

Thabet, and Bernard Ghanem. “SGAS: Sequential Greedy Architecture Search.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1620–1630, 2020.

[LSV17] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. “Hierarchical Representations for Efficient Architecture Search.”, 2017.

[LSY19a] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Autoaugment: Learning augmentation policies from data.” In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[LSY19b] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “DARTS: Differentiable Architecture Search.” In *International Conference on Learning Representations*, 2019.

[LT19] Liam Li and Ameet Talwalkar. “Random Search and Reproducibility for Neural Architecture Search.”, 2019.

[LTQ18] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. “Neural Architecture Optimization.” In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pp. 7816–7827. Curran Associates, Inc., 2018.

[Lun17] Scott Lundberg. “A unified approach to interpreting model predictions.” *arXiv preprint arXiv:1705.07874*, 2017.

[Luo22] Calvin Luo. “Understanding diffusion models: A unified perspective.” *arXiv preprint arXiv:2208.11970*, 2022.

[LYL19] Chuming Li, Xin Yuan, Chen Lin, Minghao Guo, Wei Wu, Junjie Yan, and Wanli Ouyang. "Am-lfs: Automl for loss function search." In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8410–8419, 2019.

[LZN18] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. "Progressive Neural Architecture Search." *Lecture Notes in Computer Science*, p. 19–35, 2018.

[LZS19] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. "DARTS+: Improved Differentiable Architecture Search with Early Stopping.", 2019.

[Mac98] David J. C. MacKay. "Choice of Basis for Laplace Approximation." *Machine Language*, October 1998.

[Mar18] Fritz Obermeyer Martin Jankowiak. "Pathwise Derivatives Beyond the Reparameterization Trick." In *International Conference on Machine Learning*, 2018.

[Mit09] Melanie Mitchell. *Complexity: A Guided Tour*. Oxford University Press, USA, 2009.

[MMS17a] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards deep learning models resistant to adversarial attacks." *arXiv preprint arXiv:1706.06083*, 2017.

[MMS17b] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards deep learning models resistant to adversarial attacks." *arXiv preprint arXiv:1706.06083*, 2017.

[MRK13] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. "Fine-grained visual classification of aircraft." *arXiv preprint arXiv:1306.5151*, 2013.

[MTS20] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. "Neural Architecture Search without Training.", 2020.

[MXQ19] Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. "Interpretable and steerable sequence learning via prototypes." In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 903–913, 2019.

[MZZ18] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design." In *ECCV*, 2018.

[NGH22] Weili Nie, Brandon Guo, Yujia Huang, Chaowei Xiao, Arash Vahdat, and Anima Anandkumar. "Diffusion models for adversarial purification." *arXiv preprint arXiv:2205.07460*, 2022.

[NJP21] Meike Nauta, Annemarie Jutte, Jesper Provoost, and Christin Seifert. "This looks like that, because... explaining prototypes for interpretable image recognition." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 441–456. Springer, 2021.

[NVS21] Meike Nauta, Ron Van Bree, and Christin Seifert. "Neural prototype trees for interpretable fine-grained image recognition." In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14933–14943, 2021.

[NZ08] Maria-Elena Nilsback and Andrew Zisserman. “Automated flower classification over a large number of classes.” In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pp. 722–729. IEEE, 2008.

[o22] “orhermansaffar”. “An implementation of K-Means-Auto.”, 2022.

[Ope23] OpenAI. “GPT-4 Technical Report.” *ArXiv*, **abs/2303.08774**, 2023.

[OWJ22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. “Training language models to follow instructions with human feedback.” *Advances in Neural Information Processing Systems*, **35**:27730–27744, 2022.

[Pai24] Pocket Pair. “Palworld.”, 2024.

[PD23] Dongqi Pu and Vera Demberg. “ChatGPT vs human-authored text: Insights into controllable text summarization and sentence style transfer.” *arXiv preprint arXiv:2306.07799*, 2023.

[PDS18] Vitali Petsiuk, Abir Das, and Kate Saenko. “RISE: Randomized Input Sampling for Explanation of Black-box Models.” *arXiv preprint arXiv:1806.07421*, 2018.

[PGZ18] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. “Efficient Neural Architecture Search via Parameters Sharing.” In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[PIL23] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. “Automatic prompt optimization with” gradient descent” and beam search.” *arXiv preprint arXiv:2305.03495*, 2023.

[PMS16] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. “Neuro-symbolic program synthesis.” *arXiv preprint arXiv:1611.01855*, 2016.

[RAH19] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. “Regularized Evolution for Image Classifier Architecture Search.” *Proceedings of the AAAI Conference on Artificial Intelligence*, **33**:4780–4789, Jul 2019.

[RBL22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. “High-resolution image synthesis with latent diffusion models.” In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

[RDN22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. “Hierarchical text-conditional image generation with clip latents.” *arXiv preprint arXiv:2204.06125*, **1**(2):3, 2022.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation.” In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

[RKH21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning transferable visual models from natural language supervision." In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

[RLS20] Esteban Real, Chen Liang, David So, and Quoc Le. "Automl-zero: Evolving machine learning algorithms from scratch." In *International Conference on Machine Learning*, pp. 8007–8019. PMLR, 2020.

[RMS17] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. "Large-Scale Evolution of Image Classifiers." In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 2902–2911. JMLR.org, 2017.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?: Explaining the predictions of any classifier." In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM, 2016.

[Sch44] Erwin Schrödinger. *What is Life? The Physical Aspect of the Living Cell.* Cambridge University Press, 1944.

[SCS22] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. "Photorealistic text-to-image diffusion models with deep language understanding." *Advances in Neural Information Processing Systems*, **35**:36479–36494, 2022.

[SGK17] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences." *International Conference on Machine Learning*, 2017.

[Shr14] Yu A Shreider. *The Monte Carlo method: the method of statistical trials*, volume 87. Elsevier, 2014.

[SLJ15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions." In *CVPR*, 2015.

[SM02] Kenneth O. Stanley and Risto Miikkulainen. "Evolving Neural Networks through Augmenting Topologies." *Evolutionary Computation*, **10**(2):99–127, 2002.

[SME20] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models." *arXiv preprint arXiv:2010.02502*, 2020.

[SMR23] Chandan Singh, John Morris, Alexander M Rush, Jianfeng Gao, and Yuntian Deng. "Tree prompting: Efficient task adaptation without fine-tuning." In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6253–6267, 2023.

[SRL20] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. "Autoprompt: Eliciting knowledge from language models with automatically generated prompts." *arXiv preprint arXiv:2010.15980*, 2020.

[SRM16] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. "Ladder variational autoencoders." *Advances in neural information processing systems*, **29**, 2016.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic attribution for deep networks." In *International Conference on Machine Learning*, pp. 3319–3328. PMLR, 2017.

[SWM20] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. "Hydra: Pruning adversarially robust neural networks." *Advances in Neural Information Processing Systems*, **33**:19655–19666, 2020.

[SZS20] Ameesh Shah, Eric Zhan, Jennifer Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. "Learning differentiable programs with admissible neural heuristics." *Advances in neural information processing systems*, **33**:4940–4952, 2020.

[TAB23] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. "Gemini: a family of highly capable multimodal models." *arXiv preprint arXiv:2312.11805*, 2023.

[TCP19] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. "MnasNet: Platform-Aware Neural Architecture Search for Mobile." In *CVPR*, 2019.

[TLI23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. "Llama: Open and efficient foundation language models." *arXiv preprint arXiv:2302.13971*, 2023.

[VBF15] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. "Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 595–604, 2015.

[VCC17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph attention networks." *arXiv preprint arXiv:1710.10903*, 2017.

[VWB16] Andreas Veit, Michael Wilber, and Serge Belongie. "Residual Networks Behave Like Ensembles of Relatively Shallow Networks." In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pp. 550—-558. Curran Associates, Inc., 2016.

[WA22] Sam Witteveen and Martin Andrews. "Investigating prompt engineering in diffusion models." *arXiv preprint arXiv:2211.15462*, 2022.

[WAC24] Ruochen Wang, Sohyun An, Minhao Cheng, Tianyi Zhou, Sung Ju Hwang, and Cho-Jui Hsieh. "One Prompt is not Enough: Automated Construction of a Mixture-of-Expert Prompts." In *International Conference on Machine Learning*, 2024.

[WBW11] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. "The Caltech-UCSD Birds-200-2011 Dataset." Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[WCC20] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. "Rethinking Architecture Selection in Differentiable NAS." In *International Conference on Learning Representations*, 2020.

[WCC21] Ruochen Wang, Xiangning Chen, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. "RANK-NOSH: Efficient Predictor-Based Architecture Search via Non-Uniform Successive Halving." In *ICCV*, 2021.

[WCH21] Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. "Learning fast samplers for diffusion models by differentiating through sample quality." In *International Conference on Learning Representations*, 2021.

[WDS20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. "Transformers: State-of-the-Art Natural Language Processing." In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics.

[WDZ19] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search." In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10734–10742, 2019.

[WLH24] Ruochen Wang, Ting Liu, Cho-Jui Hsieh, and Boqing Gong. "On Discrete Prompt Optimization for Diffusion Models." In *International Conference on Machine Learning*, 2024.

[WLW23] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. "Promptagent: Strategic planning with language models enables expert-level prompt optimization." *arXiv preprint arXiv:2310.16427*, 2023.

[WMM22] Zijie J Wang, Evan Montoya, David Munechika, Haoyang Yang, Benjamin Hoover, and Duen Horng Chau. "Diffusiondb: A large-scale prompt gallery dataset for text-to-image generative models." *arXiv preprint arXiv:2210.14896*, 2022.

[WNS19] Colin White, Willie Neiswanger, and Yash Savani. "BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search." *arXiv preprint arXiv:1910.11858*, 2019.

[Woo22] Max Woolf. "Lexica.", 2022.

[WRK20] Eric Wong, Leslie Rice, and J Zico Kolter. "Fast is better than free: Revisiting adversarial training." *arXiv preprint arXiv:2001.03994*, 2020.

[WSM18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." In *International Conference on Learning Representations*, 2018.

[WSY24] Ruochen Wang, Si Si, Felix Yu, Dorothea Wiesmann, Cho-Jui Hsieh, and Inderjit Dhillon. "Large Language Models are Interpretable Learners." *arXiv preprint arXiv:2406.17224*, 2024.

[WXW20] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. "Adversarial Weight Perturbation Helps Robust Generalization." In *NeurIPS*, 2020.

[WZY19] Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. "Improving adversarial robustness requires revisiting misclassified examples." In *International Conference on Learning Representations*, 2019.

[XCD22] Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Yanggang Wang, Haiyu Li, and Zhilin Yang. "GPS: Genetic prompt search for efficient few-shot learning." *arXiv preprint arXiv:2210.17041*, 2022.

[XLC22] Yuanhao Xiong, Li-Cheng Lan, Xiangning Chen, Ruochen Wang, and Cho-Jui Hsieh. "Learning to schedule learning rate with graph neural networks." In *International Conference on Learning Representation (ICLR)*, 2022.

[XXZ20] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. "PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search." In *International Conference on Learning Representations*, 2020.

[XZL18] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. "SNAS: stochastic neural architecture search." *arXiv preprint arXiv:1812.09926*, 2018.

[YWL23a] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. "Large language models as optimizers." *arXiv preprint arXiv:2309.03409*, 2023.

[YWL23b] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. "Large language models as optimizers." *arXiv preprint arXiv:2309.03409*, 2023.

[YXK22] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. "Scaling autoregressive models for content-rich text-to-image generation." *arXiv preprint arXiv:2206.10789*, **2**(3):5, 2022.

[YXT20] Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. "Efficient Neural Architecture Search via Proximal Iterations." In *AAAI*, 2020.

[YZA20] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. "Does Unsupervised Architecture Representation Learning Help Neural Architecture Search?" In *NeurIPS*, 2020.

[ZCA17] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. "Visualizing deep neural network decisions: Prediction difference analysis." *arXiv preprint arXiv:1702.04595*, 2017.

[ZCH21] Wenqing Zheng, Tianlong Chen, Ting-Kuei Hu, and Zhangyang Wang. "Symbolic Learning to Optimize: Towards Interpretability and Scalability." In *International Conference on Learning Representations*, 2021.

[ZES20] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. "Understanding and Robustifying Differentiable Architecture Search." In *International Conference on Learning Representations*, 2020.

[ZL17] Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning." In *ICLR*, 2017.

[ZLD24] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. "Benchmarking large language models for news summarization." *Transactions of the Association for Computational Linguistics*, **12**:39–57, 2024.

[ZMH22] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. "Large language models are human-level prompt engineers." *arXiv preprint arXiv:2211.01910*, 2022.

[ZSH20] Arber Zela, Julien Siems, and Frank Hutter. "NAS-BENCH-1SHOT1: BENCHMARKING AND DISSECTING ONE-SHOT NEURAL ARCHITECTURE SEARCH." In *International Conference on Learning Representations*, 2020.

[ZVS18] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. "Learning Transferable Architectures for Scalable Image Recognition." *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018.

[ZWZ22] Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E Gonzalez. "Tempera: Test-time prompting via reinforcement learning." *arXiv preprint arXiv:2211.11890*, 2022.

[ZYW18] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. "Practical Block-Wise Neural Network Architecture Generation." In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*, 2018.

[ZYW19] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. "BayesNAS: A Bayesian Approach for Neural Architecture Search." In *ICML*, pp. 7603–7613, 2019.

[ZZL18] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices." In *CVPR*, 2018.