Journal of Machine Learning Research 25 (2024) 1-41 Submitted 8/23; Revised 5/24; Published 6/24

Multi-Objective Neural Architecture Search by Learning Search Space Partitions

Yiyang Zhao YZHAO10@WPI.EDU
WORCESTER POLYTECHNIC INSTITUTE

Linnan Wang WANGNAN318@GMAIL.COM
BROWN UNIVERSITY

Tian Guo TIAN@WPI.EDU
WORCESTER POLYTECHNIC INSTITUTE

Editor: Isabelle Guyon

Abstract

Deploying deep learning models requires taking into consideration neural network metrics such as model size, inference latency, and #FLOPs, aside from inference accuracy. This results in deep learning model designers leveraging multi-objective optimization to design effective deep neural networks in multiple criteria. However, applying multi-objective optimizations to neural architecture search (NAS) is nontrivial because NAS tasks usually have a huge search space, along with a non-negligible searching cost. This requires effective multi-objective search algorithms to alleviate the GPU costs. In this work, we implement a novel multi-objectives optimizer based on a recently proposed meta-algorithm called *LaMOO* Zhao et al. (2022) on NAS tasks. In a nutshell, *LaMOO* speedups the search process by learning a model from observed samples to partition the search space and then focusing on promising regions likely to contain a subset of the Pareto frontier. Using *LaMOO*, we observe an improvement of more than 200% sample efficiency compared to Bayesian optimization and evolutionary-based multi-objective optimizers on different NAS datasets. For example, when combined with *LaMOO*, qEHVI achieves a 225% improvement in sample efficiency compared to using qEHVI alone in NasBench201. For real-world tasks, *LaMOO* achieves 97.36% accuracy with only 1.62M #Params on CIFAR10 in only 600 search samples. On ImageNet, our large model reaches 80.4% top-1 accuracy with only 522M #FLOPs.

Keywords: Neural Architecture Search, Monte Carlo Tree Search, AutoML, Deep Learning

1. Introduction

Nowadays, neural architecture search (NAS) has become instrumental in developing deep learning (DL) models that significantly surpass the performance of hand-crafted models designed by experts Zoph et al. (2018); Wang et al. (2020c); Ghiasi et al. (2019); Wang et al. (2019b,a); Real et al. (2019); Liu et al. (2019); Xu et al. (2020); Cai et al. (2020). Fundamentally, NAS aims to identify the best-performing architectures within a given search space using optimization algorithms such as reinforcement learning Zoph et al. (2018); Pham et al. (2018); Tan et al. (2019), evolutionary algorithm Real et al. (2019); Elsken et al. (2018); Lu et al. (2020); Dai et al. (2019), or Bayesian optimization Wang et al. (2019a); Liu et al. (2018); Dong et al. (2018). In real-world deployments,

metrics aside from inference accuracy are also valuable for determining a DL model's quality. For example, in face ID recognition and self-driving systems, model designers may pay more attention to the inference latency of the model. In resource-constrained devices, such as NVIDIA drive ORIN for self-driving cars ori, the designer maximizes the model accuracy while minimizing the model size/computational complexity.

As such, NAS tasks can be formulated as multi-objective optimization problems to automatically design DL models that meet all specified requirements. In this work, we explore the application of a new and effective multi-objective optimizer *LaMOO* Zhao et al. (2022) on NAS to design superior deep learning models that consider multiple, potentially conflicting, metrics. Briefly, *LaMOO* is a generic learning-based approach that effectively partitions the search space for multi-objective optimizations. Key details of *LaMOO* are presented in §4. A key question that this work seeks to answer is *how effectively* LaMOO *will perform on various multi-objective NAS tasks*.

Mathematically, in multi-objective optimization (MOO) we optimize M objectives $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x})] \in \mathrm{r}^M$:

$$\begin{aligned} \min \quad & f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_M(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} \in \Omega, \end{aligned} \tag{1}$$

where $f_i(\boldsymbol{x})$ denotes the function of objective i.

While we could set arbitrary weights for each objective to turn it into a single-objective optimization (SOO) problem, modern MOO methods aim to find the problem's entire *Pareto frontier*, the set of solutions that are not *dominated* by any other feasible solutions. Here we define *dominance* $\boldsymbol{y} \prec_{\boldsymbol{f}} \boldsymbol{x}$ as $f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y})$ for all functions f_i, and exists at least one i s.t. $f_i(\boldsymbol{x}) < f_i(\boldsymbol{y})$, $1 \leq i \leq M$. That is, if the condition holds, a solution $\boldsymbol{x}$ is always better than solution $\boldsymbol{y}$, regardless of how the M objectives are weighted. In §2.2, we will show preliminary results where single-objective optimizations fail to produce higher-quality neural architectures compared to multi-objective optimizations.

Multi-objective NAS introduces new challenges to the NAS problem. Multiple objectives bring a more complicated value space due to increasing dimensions. Finding a Pareto set of neural architectures is also more difficult than a single optimal solution. To tackle the complexity of the multi-objective NAS problem, we extend *LaMOO* to learn promising regions for NAS algorithms as will be described in §5.

Specifically, *LaMOO* learns to partition the search space into *promising* and *non-promising* regions. Each partitioned region corresponds to a node within a search tree, with the leaf nodes serving as candidates for the search process. Subsequently, *LaMOO* utilizes two variations of Monte-Carlo Tree Search (MCTS) (as will be described in §4.2.1) to select the most promising region based on the Upper Confidence Bound (UCB) values, facilitating new architecture sampling. *LaMOO* requires several initial evaluated architectures, collected via random sampling, to bootstrap the learning and search processes. *LaMOO* can be integrated with various NAS methods, including one-shot NAS, few-shot NAS, and predictor-based NAS. Details of this integration of *LaMOO* with NAS are available in §5.

Our approach, *LaMOO*, exhibits superior performance over existing methodologies across multiple NAS benchmarks, which include NAS datasets and real-world deep learning tasks. On the NasBench201 dataset, *LaMOO* boosts the sample efficiency of qEHVI and CMA-ES by 225% and 500%, respectively. Likewise, on the NasBench301 dataset, *LaMOO* enhances the sample efficiency of the original qEHVI and CMA-ES by over 200%. On the HW-NASBench dataset with four different search objectives, *LaMOO* combined with CMA-ES achieves a search performance increase

of over 250% compared to other baselines. In the context of open-domain NAS tasks, *LaMOO* also stands out. On the CIFAR-10 image classification task, *LaMOO* requires 1.5X fewer samples and identifies architectures that not only achieve similar accuracy to state-of-the-art (SoTA) models but also have fewer parameters, at 1.62M #parameters. On ImageNet, *LaMOO* found SoTA models with a top-1 accuracy of 80.4% at 522 MB #FLOPs, a top-1 accuracy of 78.0% at 248 MB #FLOPs, and a top-1 accuracy of 79.2% with only 0.57 ms TensorRT latency with FP16 on Nvidia GV100. For the MSCOCO object detection task, *LaMOO* achieves better performance at 37.6 mAP with fewer #FLOPs at 109.5G, compared to performance-oriented network ResNet-50 He et al. (2016) as the backbone.

In summary, we make the following main contributions.

- We have extended our previous work, *LaMOO* Zhao et al. (2022), to the realm of NAS problems. This study is the first to apply learning space partition in multi-objective NAS tasks. We show that *LaMOO* stands as a robust meta-optimizer capable of enhancing multi-objective NAS algorithms.
- We introduce a new search strategy called leaf selection to improve the efficiency of selecting promising regions and show that the new search strategy can improve the search efficiency for NAS problems.
- We implement *LaMOO* on different NAS datasets, including Nasbench201 Dong and Yang (2020), Nasbench301 Zela et al. (2022), and HW-NASBench Li et al. (2021). We show that using *LaMOO* can improve both Bayesian optimization and evolutionary algorithms by over 200% to 500% sample efficiency.
- *LaMOO* leads to state-of-the-art architecture performance across most real-world NAS tasks. For instance, on CIFAR-10, *LaMOO* achieves a Top-1 accuracy of 97.36% at 1.62M Parameters. On ImageNet, *LaMOO* achieves a Top-1 accuracy of 80.4% with only 522MB #FLOPs.

2. Motivation

2.1 Learning Search Space Partitions

In this section, we present a motivating example with the Branin and Currin function (Belakaria et al., 2019), demonstrating the key benefit of space partitions for multi-objective problems. The Branin-Currin is a 2-dimensional problem with two objectives. As described previously, in multi-objective problems, people usually utilize the dominance number to measure the *goodness* of the data point in existing samples Deb et al. (2002); Deb and Jain (2014). The dominance number $o(\boldsymbol{x})$ of sample $\boldsymbol{x}$ is defined as the number of samples that dominate $\boldsymbol{x}$ in search space Ω:

$$o(\boldsymbol{x}) := \sum_{\boldsymbol{x}_i \in \Omega} \mathbb{I}[\boldsymbol{x}_i \prec_{\boldsymbol{f}} \boldsymbol{x},\ \boldsymbol{x} \neq \boldsymbol{x}_i], \tag{2}$$

where $\mathbb{I}[\cdot]$ is the indicator function. This function indicates that with the decreasing of the $o(\boldsymbol{x})$, $\boldsymbol{x}$ would be approaching the Pareto frontier; $o(\boldsymbol{x}) = 0$ when sample $\boldsymbol{x}$ locates in the Pareto frontier. Figure 1 visualize the dominance number of 2000 samples in the search space of the Branin-Currin. Most of the good samples (i.e., small $o(\boldsymbol{x})$) of the search space cluster together in small regions (i.e., shaded by red).

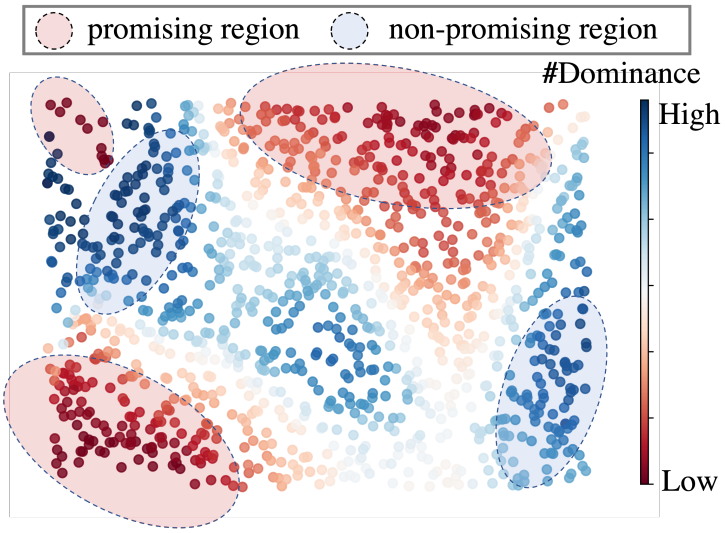


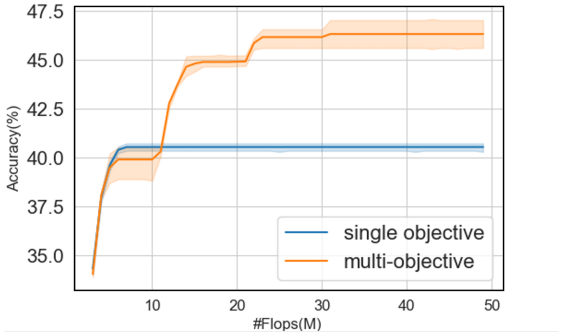
Figure 1: The visualization of search space for the Branin-Currin. Red samples have low dominance numbers, while blue ones have high dominance numbers. The smaller the dominance number, the better the quality of the sample.

This observation implies that the identification of promising regions and the subsequent concentration of optimization algorithms within these regions can significantly enhance search efficiency. Approaches based on learning space partition Wang et al. (2019a, 2020a); Zhao et al. (2022) are capable of capitalizing on these benefits. Motivated by this example, and in light of our prior work showcasing *LaMOO* as an effective multi-objective optimizer Zhao et al. (2022), our objective in this work is to examine how beneficial search space partition can be for NAS tasks. We approach neural architecture search as a multi-objective problem and apply *LaMOO*, our learning partition technique, which will be detailed in §4, to the NAS problem.

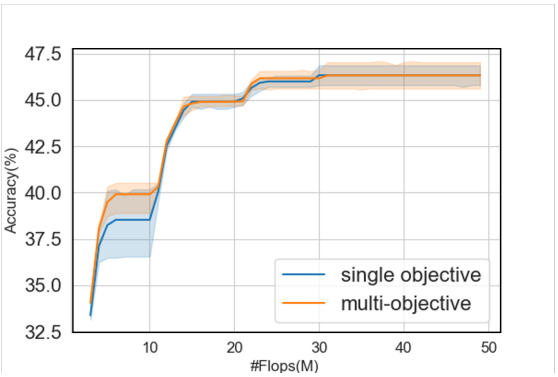
2.2 Multi-Objective Neural Architecture Search

Efficient deep neural networks concern not only traditional metrics such as accuracy, but also practical efficiency metrics, including inference latency, #FLOPs, and throughput Wu (2019). That enables efficient deep neural networks to function in limited compute capacity, limited model complexity, and limited data Wu (2019). While the design of efficient deep neural networks has drawn increased attention, most NAS works Wu et al. (2019); Wan et al. (2020); Cai et al. (2019, 2020) for efficient deep neural architectures are accuracy-oriented with constraints or by scalarizing different metrics (e.g., $\frac{accuracy}{\#FLOPs}$).

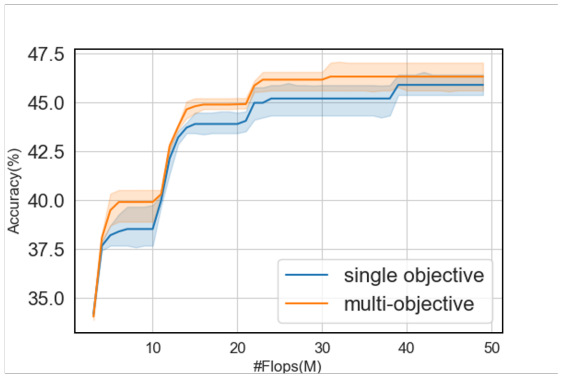
The main limitation of using constraints is that the best samples are only drawn from the constrained regions instead of the global Pareto-frontier. For example, if the #FLOPs is set to be too small, i.e., Figure 2a, the single-objective optimization-based search is often limited to architectures with low accuracy. If the #FLOPs is set to be too large, i.e., Figure. 2b, the single-objective optimization-based search concentrates on architectures of high #FLOPs. This is indicated by the single-objective optimization having significantly worse performance when the #FLOPS is less than 10M. Moreover, using the scalarized metric may deteriorate the quality of the sampled architectures. As shown in Figure. 2c, multi-objective optimization-based search remains more



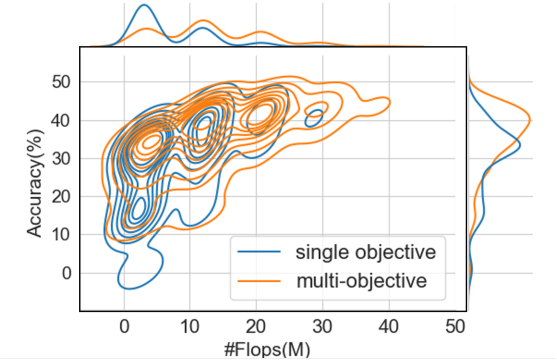
(a) constrain: $\#FLOPs < 10$M



(b) constrain: $\#FLOPs < 30$M



(c) scalarized objective $\frac{accuracy}{\#FLOPs}$



(d) contour lines with scalarized objective $\frac{accuracy}{\#FLOPs}$

Figure 2: 500 samples on Nasbench201 Dong and Yang (2020) searched by the covariance matrix adaptation evolution strategy (CMA-ES) Hansen et al. (2019); Igel et al. (2007a) with single and multi-objective versions. (a), (b) and (c) plot accuracy vs. #FLOPs of samples in the Pareto frontier. (d) is the contour lines with a sclarized objective.

effective than the single-objective counterpart because MOO better trades off any criterion in search, and Figure. 2d further points out multi-objective search cover more Pareto-set than singe-objective search. Our observations suggest that multi-objective NAS is more promising in finding efficient neural architectures with improved search efficiency.

3. Related Work

3.1 Efficient Neural Networks

Designing neural architectures to achieve the best trade-offs between performance and efficiency has emerged as a popular and important area in the deep learning community in recent years Howard et al. (2017); Sandler et al. (2018); Howard et al. (2019); Cai et al. (2020); Wu et al. (2019); Wan et al. (2020); Cai et al. (2019); Tan and Le (2019a). In particular, recent innovations focus on designing cost-efficient operations and modules. For example, MobileNetV1 Howard et al. (2017) and ShuffleNet Zhang et al. (2018) proposed the depthwise convolution and grouped convolution to reduce

the parameters and computations in the traditional convolution operation. MobileNetV2 Sandler et al. (2018) introduced a cost-friendly inverted residual block (IRB) consisting of an inverted residual and a bottleneck. More recently, the mobile-oriented MobileNetV3 Howard et al. (2019) further improves the model performance by using a new h-swish activation and a squeeze and excite module Hu et al. (2017) in the IRB. Due to its good performance, IRB has been widely used in state-of-the-art architectures Sandler et al. (2018); Howard et al. (2019); Cai et al. (2020); Dai et al. (2020) as the basic building block. Specifically, IRBs with different parameters and activation types are used to form multiple groups, serving as the key structure of the resulting models. In this work, we evaluate the efficiency of *LaMOO* in the EfficientNet search space that covers most aforementioned structures and operations. We also consider the connection pattern inside of IRB modules, which is mostly ignored by previous works. We finally compare the resulting efficient neural architectures to the state-of-the-art models.

3.2 Monte Carlo Tree Search in Neural Architecture Search

The Monte Carlo Tree Search (MCTS) algorithm is widely used in different areas, such as gaming, robotics planning, optimization, and NAS Buşoniu et al. (2013); Munos et al. (2014); Weinstein and Littman (2012); Mansley et al. (2011); Wang et al. (2019a,b). AlphaX Wang et al. (2019b) is the representative of the MCTS-based NAS algorithm. AlphaX directly leverages MCTS to search neural architectures. Each node of MCTS denotes a neural architecture, and the reward of a node is calculated by the architecture's actual performance or a value function predictor. However, MCTS-based NAS agents like AlphaX are unable to deal with multi-objective NAS requirements directly. *LaMOO* is a meta-algorithm that leverages MCTS to search the most promising regions for further sampling Zhao et al. (2022), which we apply to optimize for multi-objective NAS problems. We choose to integrate MCTS into *LaMOO* because of its effectiveness in balancing exploration and exploitation during the search process Wang et al. (2019b); Buşoniu et al. (2013); Munos et al. (2014). This integration allows *LaMOO* to explore potentially overlooked areas that might contain superior samples, even within regions initially classified as non-promising by *LaMOO*. The details of the implementation of MCTS in *LaMOO* can be found in §4.2.

Wang et al. Wang et al. (2019a) was the first to leverage MCTS and partition learning method for *single-objective* NAS problems. On top of the difference between SOO and MOO, the mechanism of the partitioning of the search space between *LaMOO* and Wang et al. (2019a) is different. Wang et al. Wang et al. (2019a) simply uses the median from the single objective of collected samples and learns a linear classifier to separate regions, while *LaMOO* leverages dominance rank and an SVM classifier to separate good regions from bad regions.

3.3 Quality Indicators of Multi-Objective Optimization

There are several quality indicators (Van Veldhuizen and Lamont, 1998; Bosman and Thierens, 2003; Zitzler et al., 2000; Bandyopadhyay et al., 2004; Jiang et al., 2014) for evaluating sample quality, which can be used to scalarize the MOO problem to the SOO problem. The performance of a quality indicator can be evaluated by three metrics (Deng et al., 2007; Li et al., 2014), including *convergence* (closeness to the Pareto frontier), *uniformity* (the extent of the samples satisfying the uniform distribution), and *spread* (the extent of the obtained samples approximate Pareto frontier). Generational Distance (GD) (Van Veldhuizen and Lamont, 1998) measures the distance between the Pareto frontier of approximation samples and the true Pareto frontier, which requires prior

Table 1: Comparison of different quality indicators.

Quality Indicator	Convergence	Uniformity	Spread	No reference set required
HyperVolume	√	√	√	√
GD	√			
IGD	√	√	√	
MS			√	
S		√		
ONVGR	√			

knowledge of the true Pareto frontier, and only considers convergence. Inverted Generational Distance (IGD) (Bosman and Thierens, 2003) is an improved version of GD. IGD calculates the distance between the points on the true Pareto frontier to the closest point on the Pareto frontier of current samples. IGD satisfies all three evaluation metrics of QI but requires a true Pareto frontier which is hard to get in real-world problems. Maximum Spread (MS) (Zitzler et al., 2000) computes the distance between the farthest two points of samples to evaluate the spread. Spacing (S) (Bandyopadhyay et al., 2004) measures how close the distribution of the Pareto frontier of samples is to uniform distribution. Overall Non-dominated Vector Generation and Ratio (ONVGR) is the ratio of the number of samples in the true Pareto frontier. In this work, we choose HyperVolume (HV) Zitzler and Thiele (1999) to evaluate the optimization performance of different algorithms because it can simultaneously satisfy the evaluation of convergence, uniformity, and spread without the knowledge of the true Pareto frontier. In addition, HV plays an important role in *LaMOO* as we leverage it to identify the goodness of partitioned spaces, from which *LaMOO* picks the best one for sampling. More details can be found in §4.1. Table 1 compares the characteristics of each quality indicator.

3.4 Search Space Optimization for Neural Architecture Search

In addition to numerous NAS studies that aim to identify the most promising architectures within a given search space, there are a number of works that focus on search space design. These studies Chen et al. (2023); Xia et al. (2022); Ci et al. (2021); Radosavovic et al. (2020b); Lin et al. (2020) seek to uncover design principles that increase the likelihood of containing more promising architectures. A notable example of this approach is found in Radosavovic et al. (2020b), where the authors designed a straightforward search space termed *AnyNet* by analyzing 500 sampled architectures to identify common traits of successful designs within these samples. However, a critical limitation of this method is that the patterns it identifies are specific to certain datasets/problems/tasks; these patterns may shift when applied to different types of problems, thus limiting its generality and applicability. In contrast, our *LaMOO* adopts a data-driven approach, systematically narrowing the entire search space into a more promising sub-region based on information collected from previously observed samples. As such, *LaMOO* can be applied to any datasets/problems/tasks. We will show that the searched models by *LaMOO* outperform the ones by Radosavovic et al. (2020b) in Table. 5.

MCUNet Lin et al. (2020) offers an alternative method for optimizing the search space in NAS, tailored specifically for neural network architectures that must operate within the constraints of certain devices, such as microcontrollers with limited memory or specific latency requirements. Specifically, MCUNet predefines a variety of search spaces based on different input resolutions and width multipliers. The underlying assumption is that models with greater computational requirements

Table 2: Notation definitions through the paper.

Ω	the whole architecture space	a	an architecture in the architecture space	$o(a)$	dominance number of architecture a
Ω_j	the partition of Ω represented by the tree node j	n_i	number of samples in node i	$D_t \cap \Omega_j$	samples in node j
D_t	samples in iteration t	v_i	the multiple evaluation metrics of a_i	H_j	Hypervolume of $D_t \cap \Omega_j$

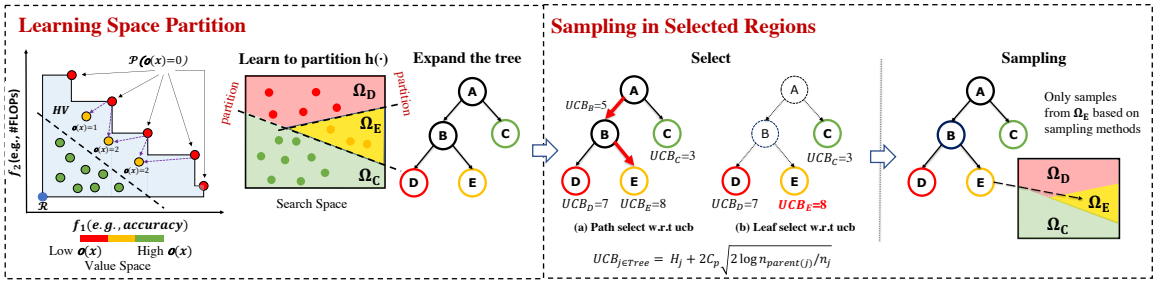


Figure 3: **The overview of a *LaMOO* iteration.** The left portion depicts the *Learning Space Partition* phase for optimizing two objectives. The first figure depicts the value space (f_1, f_2) and visualizes the hypervolume HV (blue-shaded area) given the Pareto frontier P and the reference point R. The middle figure shows the search space Ω and its partitions (i.e., Ω_C, Ω_D, and Ω_E) based on samples collected from the previous iterations and their dominance numbers in the objective space. The right figure shows the tree constructed based on the partitions. The right portion depicts the *Sampling in Selected Regions* phase. The left figure visualizes two selection strategies described in §4.2. The right figure shows that new architectures will be sampled from the good partition Ω_E with any sampling algorithms. Figure adapted from our prior work Zhao et al. (2022).

have a larger capacity and are, therefore, more likely to achieve higher accuracy. Based on this assumption, MCUNet randomly samples 1000 architectures from each of the predefined search spaces, subsequently selecting those that meet the specific requirements of the target devices (such as memory capacity and latency). By calculating the average #FLOPs for the architectures that fulfill these criteria within each search space, they identify search spaces with higher average #FLOPs that have the potential to yield promising architectures. This method, however, has two limitations. First, it may not be applicable if the architectural design is constrained by requirements related to #FLOPs. Second, it requires additional domain knowledge and human effort to design the candidate search spaces. Instead of focusing solely on #FLOPs, our *LaMOO* can work with multiple metrics (e.g., #Params, #FLOPs, latency, accuracy). It utilizes data from previously evaluated architectures to partition the search space and identify promising regions that are more likely to contain architectures meeting the design requirements. *LaMOO* is a parallel approach to MCUNet Lin et al. (2020), *LaMOO* can complement it by further refining the search space based on historical architecture samples. When MCUNet is used to select an appropriate initial search space for a given target device, *LaMOO* can then refine this space to focus on promising sub-regions.

4. Multi-Objective Optimization by Learning Space Partition

In this section, we present the key details of the learning space partitions based on our previously proposed multi-objective algorithm, referred to as *LaMOO* Zhao et al. (2022). Briefly, *LaMOO* is a *meta-optimization algorithm* that separates good regions out from the entire search space by using

Algorithm 1 Pseudo-code of *LaMOO* for the NAS task.

Inputs: Initial D_0 from uniform sampling, sample budget T.
for $t = 0, \ldots, T$ **do**
  Set $\mathcal{L} \leftarrow \{\Omega_{\text{root}}\}$ (collections of regions to be split).
  while $\mathcal{L} \neq \emptyset$ **do**
    $\Omega_j \leftarrow \text{pop\_first\_element}(\mathcal{L})$, $D_{t,j} \leftarrow D_t \cap \Omega_j$, $n_{t,j} \leftarrow |D_{t,j}|$.
    Compute dominance number $o_{t,j}$ of $D_{t,j}$ using Eqn. 2 and train a SVM model $h(\cdot)$.
    If $(D_{t,j}, o_{t,j})$ is splittable by SVM, **then** $\mathcal{L} \leftarrow \mathcal{L} \cup \text{Partition}(\Omega_j, h(\cdot))$.
  end while
  if *Path Selection* **then**
    for $k = \text{root}$, k is not leaf node **do**
      $D_{t,k} \leftarrow D_t \cap \Omega_k$, $v_{t,k} \leftarrow \text{HyperVolume}(D_{t,k})$, $n_{t,k} \leftarrow |D_{t,k}|$.
      $k \leftarrow \arg \max_{c \in \text{children}(k)} \text{UCB}_{t,c}$, where $\text{UCB}_{t,c} := v_{t,c} + 2C_p\sqrt{\frac{2\log(n_{t,k}}{n_{t,c}}}$
    end for
  end if
  if *Leaf Selection* **then**
    for $k = \text{root}$, k is not leaf node **do**
      $D_{t,k} \leftarrow D_t \cap \Omega_k$, $n_{t,k} \leftarrow |D_{t,k}|$.
    end for
  end if
  for l is leaf node **do**
    $v_{t,l} \leftarrow \text{HyperVolume}(D_{t,l})$
  end for
  $k \leftarrow \arg \max_{l \in \text{leaf nodes}} \text{UCB}_{t,l}$, where $\text{UCB}_{t,l} := v_{t,l} + 2C_p\sqrt{\frac{2\log(n_{t,l})}{n_{t,p}}}$, where p is the parent of l.
  $D_{t+1} \leftarrow D_t \cup D_{\text{new}}$, where D_{new} is drawn from Ω_k based on sampling algorithms such as qEHVI or CMA-ES.
end for

observed data. Different multi-objective search algorithms, such as qEHVI Daulton et al. (2020), CMA-ES Igel et al. (2007), and random sampling, can be combined with *LaMOO* in these promising regions for sampling. In this paper, we also introduce a novel promising region selection method, i.e., leaf selection in §4.2.1. Table 2 lists notations that are used throughout the paper.

Similar to our prior work, LaNAS Wang et al. (2019a) and LaMCTS Wang et al. (2020a), an iteration of *LaMOO* consists of the learning and sampling phases. *LaMOO* iterates between learning space partition (§4.1) and Monte Carlo Tree Search (§4.2) until depleting the sample budget T, which can be either search time or the number of samples. Figure 3 presents an overview of an iteration in *LaMOO*. The pseudo-code is presented in Algorithm 1.

4.1 Learning Partitions in Multi-Objective Search Space

4.1.1 OVERVIEW OF THE PARTITION LEARNING ALGORITHM

The learning phase of iteration t begins with D_t, a dataset consisting of tuples (a_i, v_i), which represent previously observed samples. Here, a_i symbolizes a single architecture represented by an encoding vector. Note that different search spaces have different encoding representations (e.g., variations in vector length and range). Moreover, v_i represents the evaluation metrics for the corresponding

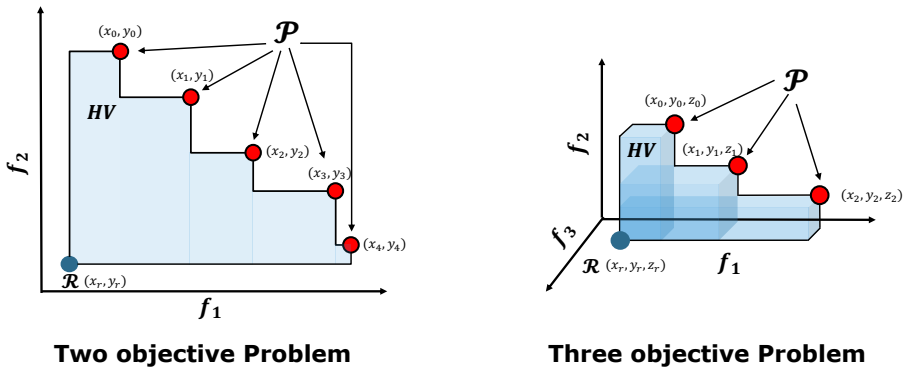


Figure 4: Visualization of example hypervolume calculation.

architecture a_i. Typically, for multi-objective NAS, v_i can include the architecture performance metrics (e.g., accuracy) and inference-phase metrics (e.g., #params, #FLOPs, and latency).

For the first iteration, we initialize D_t with a few randomly sampled architectures. The performance metrics v_i can be obtained in the following three main ways with different data qualities and acquisition costs: *(i)* actual training results Zoph et al. (2018); Wang et al. (2019b); Real et al. (2019); *(ii)* pre-trained dataset results, e.g., NasBench201 Dong and Yang (2020); Ying et al. (2019); *(iii)* supernet-based estimated results Zhao et al. (2021); Bender et al. (2018). We describe how *LaMOO* leverages different v_i representations to search multi-objective architectures in §5.

During any iteration of the search process, our objective is to identify promising regions (for example, the red region in Figure 1) from the search space Ω and then concentrate the search action on these promising regions. To achieve this, we construct a search tree that recursively partitions the search space into *good* and *bad* regions, thereby learning the partitions and identifying the optimal region for the search.

At iteration t, with observed samples D_t and a root node representing the entire search space Ω_{root} ($\Omega_{root} = \Omega$), we recursively partition the search space from the root node to the leaves. Specifically, at node j, we partition the current search space Ω_j into two disjoint regions, Ω_{good} and Ω_{bad}, where $\Omega_j = \Omega_{good} \cup \Omega_{bad}$. The Ω_{good} and Ω_{bad} regions are partitioned based on the rank of the *dominance number* of D_t. The implementation details are provided in §4.1.2. We quantify the *goodness* of a search space using the metric *hypervolume* (HV) of samples in the space. A larger hypervolume value signifies a more promising space. The definition of HV is provided below.

Given a reference point $R \in \mathrm{r}^M$ (e.g., as shown in Figure 4), the *hypervolume* of a finite approximate Pareto set $\mathcal{P}$ is the M-dimensional Lebesgue measure λ_M of the space dominated by $\mathcal{P}$ and bounded from below by R. That is, $HV(\mathcal{P}, R) = \lambda_M(\cup_{i=1}^{|\mathcal{P}|}[R, y_i])$, where $[R, y_i]$ denotes the hyper-rectangle bounded by reference point R and y_i. We present visualizations for two and three-objective optimization examples in Figure 4. The hypervolume is indicated by the blue-shaded area. For a two-objective problem, the hypervolume $HV(\mathcal{P}, R)$ is calculated as $\sum_{i=1}^{n-1}(x_i - x_{i-1}) \cdot (y_i - y_r) + (x_0 - x_r) \cdot (y_0 - y_r)$. The hypervolume computation for a three-objective problem is more complex. Considering a simplified scenario where $(x_0 < x_1 < x_2) \wedge (y_0 > y_1 > y_2) \wedge (z_0 > z_1 > z_2)$, the hypervolume $HV(\mathcal{P}, R)$ is given by $(x_0 - x_r) \cdot (y_0 - y_r) \cdot (z_0 - z_r) + (x_1 - x_0) \cdot (y_1 - y_r) \cdot (z_1 - z_r) + (x_2 - x_1) \cdot (y_2 - y_r) \cdot (z_2 - z_r)$.

The resulting partitions then satisfy the following property of $H_{good} > H_{bad}$, where H_{good} is the hypervolume calculated based on $D_t \cap \Omega_{good}$ and H_{bad} is based on $D_t \cap \Omega_{bad}$. By repeating the partitioning process, we can construct a tree comprising nodes that partition the entire search space into diverse performance regions, in terms of hypervolume, for multi-objective optimization. The following illustrates the node-level partitioning in detail.

4.1.2 DETAILS OF NODE-LEVEL PARTITIONING

At each node j, for each sampled architecture $\boldsymbol{a} \in D_t \cap \Omega_j$, we calculate its *dominance number* $o_{t,j}(\boldsymbol{a})$, defined in equation 2, to represent its performance. To speed up the calculation, we use Maxima Set (Kung et al., 1975) which runs in $O(|D_{t,j}| \log |D_{t,j}|)$, compared to naive computation which requires $O(|D_{t,j}|^2)$ operations. Note that architectures in the Pareto frontier have a dominance number of 0. In other words, the lower the dominance number, the better the architecture. We, therefore, sort all sampled architectures ($D_t \cap \Omega_j$) based on their dominance numbers in descending order and label the first half as good samples and the remaining as bad samples.

After all samples are labeled in node j, we construct a SVM classifier $h(\cdot)$ fitting the labeled architectures $c(\boldsymbol{a})$ as below:

$$\underset{a_i \in D_t \cap \Omega_j}{min} \sum_i (h(a_i) \oplus c(a_i)). \tag{3}$$

Given that we have categorized all sampled architectures ($D_t \cap \Omega_j$) as either good or bad, we formulate the task of partitioning the space as a binary classification problem. In this scenario, $h(a_i), c(a_i) \in \{0, 1\}$, with $\oplus$ symbolizing the XOR operation. Within this framework, 0 signifies a bad architecture, while 1 indicates a good one. This implies that $c(a_i) = 1$ denotes a good architecture, while $c(a_i) = 0$ indicates a bad architecture. Equation 3 is equivalent to $h(a_i) \cdot (1 - c(a_i)) + c(a_i) \cdot (1 - h(a_i))$. Consequently, the minimum of the above equation is 0 if all samples are classified correctly. In the worst-case scenario, where all samples are misclassified, the value equates to the number of samples.

Upon completion of the classifier training, the search space Ω_J bifurcates to a good and a bad region (i.e., Ω_{good} and Ω_{bad}) by the $h(\cdot)$. For ease of exposition, we designate the left child of node j as the good region and the right child as the bad region of Ω_j.

The search space can be continuously partitioned, and the corresponding search tree can be constructed by repeating the aforementioned steps until one of the stopping conditions is met. We consider the following three conditions: *(i)* If the samples in a node cannot be split by $h(\cdot)$, this node will be marked as a leaf node. Here, a node is considered non-splittable if all samples receive the same label or if the classifier cannot classify based on the current samples (i.e., always predict to 0 or 1). *(ii)* If the tree reaches the maximum height, which is considered a hyper-parameter in this work, the process will halt. *(iii)* If the number of samples in a node is less than the *minimal sample threshold*, which is also a hyper-parameter, this node will be marked as a leaf node.

Once the entire space is partitioned (i.e., the search tree is constructed), partitions represented by leaf nodes follow the sequence $H_{leftmost} > \ldots > H_{rightmost}$, with the leftmost leaf node representing the most promising partition. A detailed visualization of the space partitioning and its effectiveness is shown in Figure 11.

4.2 Sampling from the Promising Region with MCTS

Once the space partitions are learned as previously described, the next step is to search the constructed tree. The primary goal is to sample the most promising neural architectures from the selected regions and use the obtained $(\boldsymbol{a}_i, \boldsymbol{v}_i)$ tuples to update the learning phase for the next iteration. We employ the Monte Carlo Tree Search (MCTS) algorithm to explore and exploit the learned partitions, thereby preventing us from getting stuck in local optima Wang et al. (2019a). Similar to the traditional MCTS algorithm, our LaMOO search incorporates *selection*, *sampling*, and *backpropagation* stages. We omit the expansion part of MCTS because our search tree, including the structure and the learned classifiers, will remain fixed during the search phase. We elaborate on two strategies for selecting promising regions in §4.2.1 and suitable sampling methods in §4.2.2.

4.2.1 PROMISING REGION SELECTION STRATEGIES

In this work, we consider two selection strategies for implementing the MCTS. The *path selection* strategy derives from the original MCTS algorithm, while the *leaf selection* strategy is a computation-efficient variation that saves Hypervolume calculation. Figure 11 visualizes the resultant partitions and Pareto frontier with the path selection strategy.

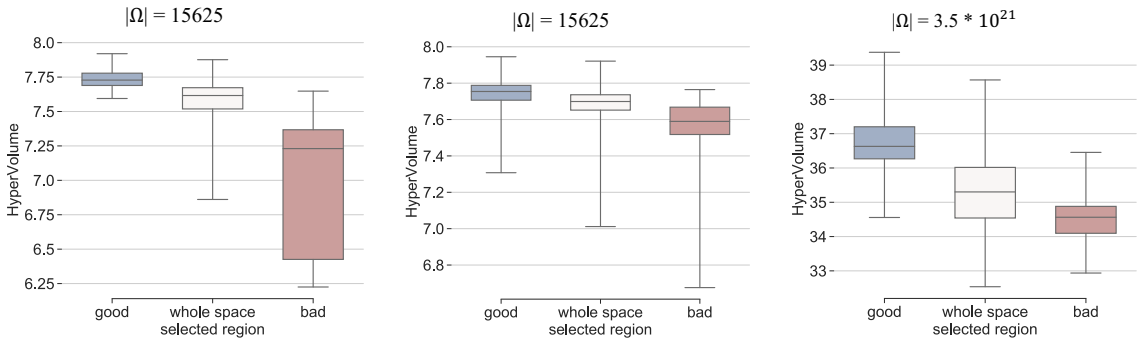
Path selection works by traversing down the constructed tree to generate the most promising path. The search starts from the root node and stops when a leaf node is reached. At a given node, the agent determines which child node to traverse based on the UCB1 value Auer et al. (2002). The UCB1 value of node j is defined as:

$$\mathrm{UCB}_j := H_j + 2C\sqrt{\frac{2\log n_{\mathrm{parent(j)}}}{n_j}}. \tag{4}$$

In this equation, n_j denotes the number of samples in node j, $n_{\mathrm{parent(j)}}$ refers to the number of samples in node j's parent, C is a tunable hyperparameter that adjusts the degree of exploration, and H_j denotes the hypervolume of the samples in node j. The first term H_j is the exploitation term which evaluates the expected multi-objectives performance of samples in the current node j. The second term, $2C\sqrt{\frac{2\log n_{\mathrm{parent(j)}}}{n_j}}$, represents the exploration term that encourages the selection of the next node with fewer samples. To summarize, given two sibling nodes i and j, we will traverse node i if $UCB_i \geq UCB_j$. Hence, the path selection strategy ensures all nodes in the chosen path possess a larger UCB value compared to their sibling node. However, this search strategy demands the computation of the hypervolume for every node, which can be highly resource-intensive. Prior work Beume and Rudolph (2006) shows that when the number of objectives M is more than three, the computation cost of hypervolume is $O(N^{\frac{M}{2}} + N\log N)$, where N is the number of searched samples in total[1]. That is, the hypervolume computation cost is growing exponentially with M when $M > 3$. Next, we describe the *leaf selection* strategy, which mitigates this high computation cost problem.

Leaf selection significantly reduces the hypervolume computation by only calculating the UCB1 value for all the leaf nodes. The leaf node with the highest UCB1 value is selected as the promising region. This strategy avoids computing the hypervolume in the non-leaf nodes of the tree, where hypervolume calculation is the primary computational cost of *LaMOO*, especially in many-objective problems (i.e., $M > 3$). Moreover, even if the number of objectives is less than three, our leaf

1. When $M \leq 3$, the computation complexity of hypervolume is $O(N\log N)$.



(a) NasBench201 (b) NasBench201 with supernet (c) NasBench301

Figure 5: The range of hypervolume for 50 samples randomly generated from different regions in NasBench201, NasBench201 by supernet search, and NasBench301.

selection strategy can still save computational time compared to the path selection strategy. This is because, in this case, the number of samples becomes the dominant factor in hypervolume computation. Generally, the number of samples in the leaf nodes is fewer than in the non-leaf nodes, especially nodes nearer to the root. Figure 12d shows that the node selection strategy can achieve a similar sample performance with less time compared to the path selection strategy.

4.2.2 SAMPLING FROM THE PROMISING LEAF NODE

After selecting the promising leaf node j, we can utilize multi-objective sampling methods such as random sampling, qEHVI (Daulton et al., 2020), and CMA-ES (Igel et al., 2007a) to select samples. Therefore, we regard *LaMOO* as a *meta-algorithm* that can enhance the search performance of existing MOO algorithms.

Random sampling. We implement the *reject sampling* technique in which we randomly sample the promising partition Ω_j and only keep samples that satisfy the constraint. Here, we consider each $h(\cdot)$ in the nodes as a constraint and only select samples that are classified as good in the selected path. To understand the effectiveness of *LaMOO* even with random sampling, we conduct a preliminary evaluation as follows. Specifically, we randomly generate ten samples from NasBench201 or NasBench301 as the initialization. In each iteration, we update 5 batched samples for all search algorithms and 300 samples in total. The setup of *LaMOO* followed Zhao et al. (2022). We run *LaMOO* 5 times and run the random sampling for each region 150 times. We also evaluate the effectiveness of learning partitions in one-shot NAS on Nasbench201, labeled as *Nasbench201 with supernet*. We train a supernet based on Zhao et al. (2021) and leverage the supernet to estimate the accuracy of sampled architecture. All the search and sampling processes are based on the results from the supernet. Figure 5 compares the random sampling performance where each box plot denotes the hypervolume distribution with randomly generated 50 samples in different selected regions, i.e., good, whole space, and bad. We observe that searching in the good region for NasBench201 improves the median hypervolume and leads to a tighter distribution compared to searching in the entire space, as shown in Figure 5a. Similar trends are observed for NasBench301 as shown in Figure 5c. Moreover, we see that compared to standard NAS, one-shot NAS (Figure 5b has a worse

hypervolume distribution in the good region but better results in the bad region[2]. This is largely in part because the estimation by the supernet is not accurate as claimed by prior work Bender et al. (2018), which results in inaccurately learned partitions as well.

Bayesian optimization. A typical Bayesian Optimization for NAS works by first training a surrogate model using a Gaussian Process Regressor (GPR) on observed architectures and then generating new architectures based on the acquisition function, such as Expected Improvement (EI) or Upper Confidence Bound (UCB) Jones et al. (1998); Auer et al. (2002). In this work, we integrate *LaMOO* with a state-of-the-art multi-objective Bayesian Optimization solver named qEHVI (Daulton et al., 2020), which finds samples to optimize parallel version of the acquisition function called Expected Hypervolume Improvement (EHVI). To incorporate qEHVI into *LaMOO*'s sampling step, we confine qEHVI's search space to a sub-space represented as a node in the MCTS tree. As claimed in our previous space partition work Zhao et al. (2022), *LaMOO* leverages previous samples to learn the partitions, which convert complicated non-convex optimization of the acquisition functions in Bayesian optimization into a simple traversal of hierarchical partition tree while still precisely capturing the promising regions for the sample proposal, reducing the complexity of Bayesian Optimization.

Evolutionary algorithms. A typical evolutionary algorithm (EA) search step consists of two parts. The first part is *selection*, which chooses several samples with the best performance. The second part is *mutation*, which slightly alters the selected samples to propose new individuals Igel et al. (2007a); Real et al. (2019). Take the NAS task as an example. The mutation part changes 1-2 connections or existing operations of the selected best architectures to evolve the entire population in the current step Real et al. (2019). In this work, we combine *LaMOO* with CMA-ES (Igel et al., 2007a), an evolutionary algorithm originally designed for multi-objective optimization. To apply CMA-ES in the multi-objective NAS, we use CMA-ES to sample the promising region Ω_j by picking an architecture that minimizes the dominance number $o(\boldsymbol{x})$. Note that $o(\boldsymbol{x})$ changes over iterations, so we need to update $o(\boldsymbol{x})$ with new samples before running CMA-ES.

4.2.3 OPTIONAL BACKPROPAGATE REWARDS

A distinctive trait of MCTS is giving more preference to exploration by backpropagation. Backpropagation backtracks the selected path from the leaf to the root, updating the visit counts and values of the samples. Specifically, for node j on the selected path, the visit count n_j is updated with the number of new samples, and so is the hypervolume H_j. This backpropagation step assists in evolving our search tree from one iteration to the next. However, this step can be bypassed by directly reconstructing the tree using the newly generated samples. We use this non-backpropagation version in our experiments throughout the paper.

5. Multi-Objective Learning Space Partitions with Different NAS Methods

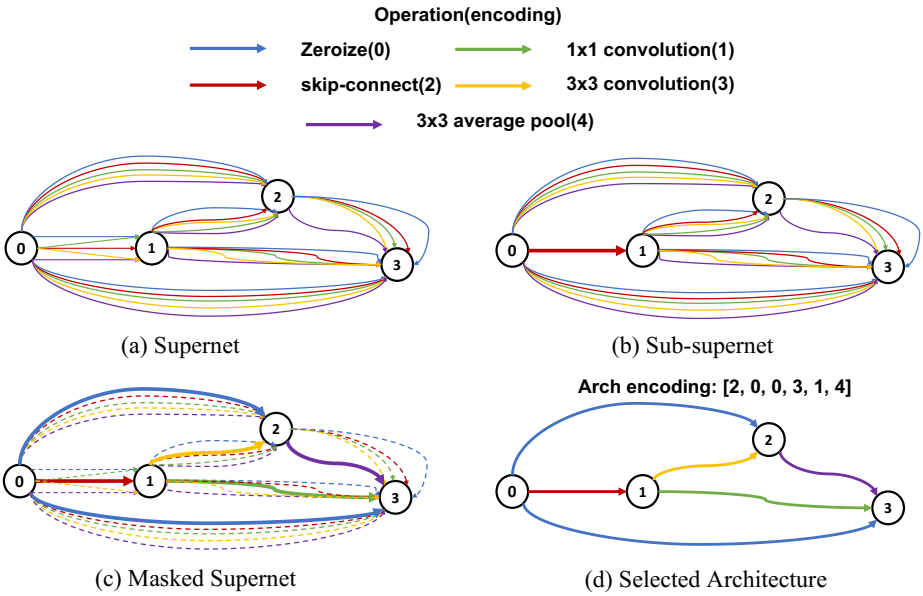
In this section, our goal is to demonstrate the efficacy of *LaMOO* when applied to the problem of *multi-objective neural architecture search*. We detail how *LaMOO* can be combined with three major NAS evaluation approaches to enhance search efficiency. These approaches differ in how we obtain the architecture and its performance metrics (a_i, v_i), and the associated evaluation costs. Table 3 compares popular NAS algorithms from multiple perspectives. Our *LaMOO* is the only NAS method

2. Note that the hypervolume calculation is based on true accuracy in NasBench201.

Table 3: Comparisons of existing NAS methods.

NAS method	Search method	One(few)-shot support	Performance prediction	Multi-objective	Meta-optimizer
NASNet Zoph et al. (2018)	Reinforcement learning				
ENAS Pham et al. (2018)	Reinforcement learning	√			
MnasNet Tan et al. (2019)	Reinforcement learning			√†	
AmoebaNet Real et al. (2019)	Evolutionary algorithm				
LEMONADE Elsken et al. (2018)	Evolutionary algorithm		√	√	
NSGANetV2 Lu et al. (2020)	Evolutionary algorithm	√	√	√	
ChamNet Dai et al. (2019)	Evolutionary algorithm	√		√†	
OFANet Cai et al. (2020)	Evolutionary algorithm	√		√†	
PNAS Liu et al. (2018)	SMBO‡		√		
DPP-Net Dong et al. (2018)	SMBO‡		√	√†	
DARTS Liu et al. (2019)	Gradient	√			
PCDARTS Xu et al. (2020)	Gradient	√			
FBNetV2 Wan et al. (2020)	Gradient	√		√†	
LaNAS Wang et al. (2019a)	Space partition	√			√
LaMOO (ours)	Space partition	√	√	√	√

† Optimization leverages scalarized multiple objectives (e.g., objective in Figure 2c or additional constrains (e.g., objective in Figure 2a&Figure 2b).
‡ Sequential Model Based Optimizations.



(a) Supernet (b) Sub-supernet

(c) Masked Supernet (d) Selected Architecture

Figure 6: The cell structure of the supernet used in Nasbench201. (a) is a supernet structure of the NasBench201 cell. Any edge between two nodes is combined with four operations. (b) is a sub-supernet of the NasBench201 cell. The edge between node 0 and node 1 has only one operation. (c) demonstrates a masked supernet that only one path is activated. The dotted lines are inactive edges. (d) is a cell architecture example from NasBench201 with its encoding.

supporting all distinct NAS evaluation approaches (one-shot Bender et al. (2018); Yu et al. (2019b); Zhao et al. (2021); Pham et al. (2018), few-shot Zhao et al. (2021), and performance prediction Liu et al. (2018)); moreover, *LaMOO* is compatible with multi-objective optimization and can act as a meta-optimizer to enhance other NAS methods.

5.1 One-Shot Neural Architecture Search

Prior work proposed to mitigate the high evaluation cost in vanilla NAS Zoph et al. (2018); Real et al. (2019); Wang et al. (2019b) with a weight-sharing technique, avoiding retraining sampled networks from scratch Pham et al. (2018). This type of NAS is referred to as *one-shot neural architecture search*. Typically, one-shot NAS first trains an over-parameterized *supernet* that covers all potential operations and connections of the entire search space and searches on the supernet as an integrated bi-level optimizations Liu et al. (2019); Xu et al. (2020). Figure 6(a) shows an example of the supernet topology used for the NasBench201 search space.

Besides the bi-level optimization approach Liu et al. (2019), another variant of one-shot NAS uses the supernet as a performance estimator to predict the architecture's performance Bender et al. (2018); Yu et al. (2019b); Zhao et al. (2021); Pham et al. (2018). The performance of any architecture from the search space can be estimated using a well-trained supernet by sharing common weights.

Our *LaMOO* integrates with the latter variant of one-shot NAS, which efficiently assesses the performance of sampled architectures (e.g., accuracy). Note that due to co-adaptation among operations Bender et al. (2018), the performance assessed by the supernet is not as accurate as the performance obtained through actual training of the sampled architectures. §5.2 elaborates on few-shot NAS Zhao et al. (2021), which enhances the evaluation performance. The following describes the supernet design, its training, and its usage in detail.

Supernet design. We use the following three supernet designs in this work.

- *NasBench201* Dong and Yang (2020). Our supernet design follows Yu et al. (2019a); Zhao et al. (2021). Figure 6(a) shows the topology of the supernet in the NasBench201 search space. The supernet keeps the same nodes as the architectures in the search space. However, each node in the supernet is connected by all possible operations with weight addition. Figure 6(d) illustrates an architecture that includes four feature map nodes and six operation edges.

- *DARTS search space on CIFAR10* Liu et al. (2019) . On the CIFAR10 classification task, we use the same supernet design as DARTS Liu et al. (2019). The network architecture is built by stacking multiple normal cells and reduction cells. We only perform NAS on the structure inside of the cells. The normal cell keeps the same dimensions as the input feature map, while the reduction cell cuts the height and width of the feature map by two and multiplies the channel numbers by two. The supernet consists of four nodes, which are connected by all available candidate operations. We replace the bi-optimization search process on DARTS with the random mask training because we only use the supernet as a performance estimator for *LaMOO*.

- *EfficientNet search space on ImageNet* Tan and Le (2019a). On the ImageNet classification task, we follow the supernet design by Cai et al. (2020). The once-for-all supernet supports predicting different architectures with different depths, widths, kernel sizes, and resolutions without retraining.

Supernet training. In this work, we leverage the random mask strategy Guo et al. (2019) to train a supernet to convergence. This is because this training strategy was shown to enhance the supernet evaluation performance regarding rank co-relation Guo et al. (2019); Zhao et al. (2021). Specifically, instead of training the entire supernet in each iteration, we randomly pick one architecture from the search space, train it for one epoch, and update the corresponding weights in the supernet.

Architecture evaluation by supernet. After training the supernet, any architecture from the search space can be evaluated by the supernet. For evaluating a specific architecture like the one shown in Figure 6(d), we first mask out all other unused edges (dotted line as shown in Figure 6(c)), and use the supernet with the remaining weights to estimate the architecture performance.

5.2 Few-shot Neural Architecture Search

While one-shot supernet enables quick estimation of architectures in the search space, many works Yu et al. (2019b); Zhao et al. (2021); Luo et al. (2018); Dong and Yang (2020); Luo et al. (2020); Bender et al. (2018) show that supernet often leads to search performance degradation due to inaccurate architecture performance prediction. To address this downside of supernet, Zhao et al. (2021) proposed few-shot NAS which greatly reduces the negative impact of co-adaptation among operations. Specifically, instead of using one supernet, the search space is represented with multiple sub-supernet, each covering a part of the search space. Each sub-supernet, therefore, uses fewer operations in a compound edge. Figure 6(b) is an example of a sub-supernet from the same search space. The compound edge between node 0 and node 1 is simplified to one operation.

In short, because prior work demonstrates that few-shot NAS can greatly improve the search performance compared to the one-shot NAS Zhao et al. (2021); Hu et al. (2022); Xu et al. (2022); He et al. (2022); Su et al. (2021), we also integrate *LaMOO* with few-shot NAS in this work. The training strategy of few-shot NAS is similar to that of one-shot NAS. To speed up the training procedure of sub-supernets, few-shot NAS leverages the weight-transferring technique. Specifically, a supernet is trained from scratch until convergence. All sub-supernet training then directly leverage the weights from the supernet and only take a few epochs to converge. This training strategy can save much training time compared to training all sub-supernets from scratch.

The architecture evaluation is similar to the steps described above for one-shot NAS. We first pick the corresponding sub-supernet and mask the unused operations in the sub-supernet to estimate the architectures' performance.

5.3 Performance Predictor Guided Search

Besides supernet, another efficient NAS approach is to train performance predictors (e.g., deep neural network) based on observed architectures to guide the search process Liu et al. (2018); Wang et al. (2019b); Dong et al. (2018). The accuracy of such performance predictors, in part, depends on the information about architectures, e.g., the number of trained architectures. In this work, we integrate the performance predictor from NasBench301, which was demonstrated to have good prediction accuracy Zela et al. (2022), with *LaMOO*. This predictor is trained on 60K architectures.

6. Experiments

We evaluate our *LaMOO* algorithm on two types of NAS scenarios. The first type is based on three popular NAS datasets, NasBench201 Dong and Yang (2020), NasBench301 Zela et al. (2022) and HW-NAS-Bench Li et al. (2021). The second type is real-world deep learning domain applications, including image classification, object detection, and language models.

6.1 NAS Datasets

To date, there are three popular open-source NAS datasets, NasBench101 Ying et al. (2019), NasBench201 Dong and Yang (2020), and NasBench301 Zela et al. (2022). For our evaluation, we chose the latter two because the network architectures in these two datasets cover the entire search space, while the NasBench101 dataset only consists of a small subset of architectures. Evaluating using NasBench101 is challenging because we will not have access to important information, such as accuracy, during the search. We also use HW-NAS-Bench Li et al. (2021), a hardware-aware neural architecture search benchmark, which offers extensive metrics of architectures in NasBench201 Dong and Yang (2020) for many-objective NAS.

For each NAS dataset, we evaluate the search performance of using *LaMOO* in conjunction with two SoTA algorithms called qEHVI Daulton et al. (2020) and CMAES Igel et al. (2007a). Specifically, qEHVI and CMA-ES are Bayesian optimization (BO)-based and evolutionary-based multi-objective optimization algorithms, respectively. The other search algorithms we evaluated include qPAREGO Knowles (2006) (BO-based), NSGA-family Deb et al. (2002); Deb and Jain (2014) (evolutionary-based), and MOEAD Zhang and Li (2007) (evolutionary-based).

6.1.1 EVALUATION USING NASBENCH201

Dataset overview. NasBench201 is an open-source benchmark and dataset for evaluating NAS algorithms Dong and Yang (2020). In Nasbench201, the architectures are formed by stacking the cells together. Figure 6(a) depicts the design of a cell as a fully-connected graph. Specifically, each cell contains 4 nodes and 6 edges. Each node is a feature map, and the edge represents a type of operation. A pair of nodes are connected by one of the following operations, i.e., zeroize, skip-connect, 1x1 convolution, 3x3 convolution, and 3x3 average pooling. To represent each architecture uniquely, we encoded the five operations with the numbers 0 to 4 and used a 6-length vector to represent a specific architecture.

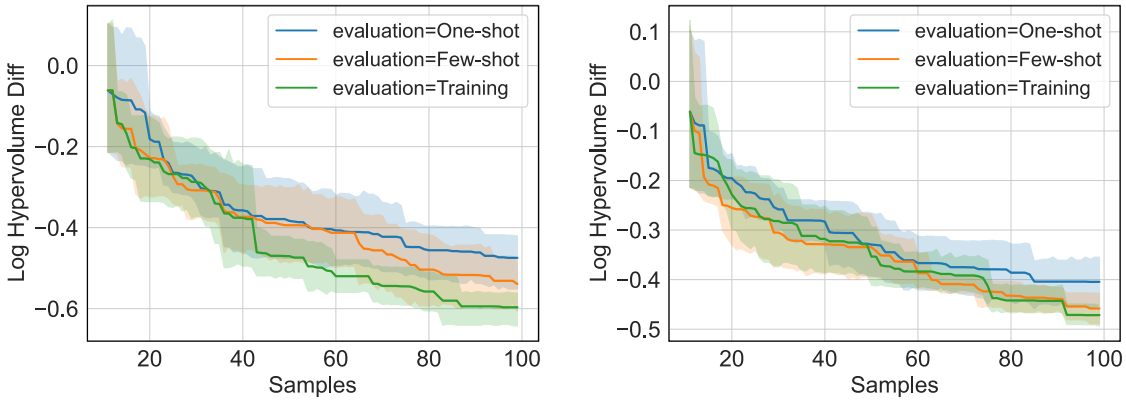
We chose two objectives, #FLOPs and accuracy, to optimize. We normalized #FLOPs to range $[-1, 0]$ and accuracy to $[0, 1]$. NasBench201 provides all architectures' information in its search space and comprises 15625 architectures trained to converge on CIFAR10 Krizhevsky (2009). As such, NAS algorithms can leverage the preexisting information about each architecture's #FLOPs and accuracy as ground truth to avoid time-consuming training during algorithm evaluation. After the normalization, we also calculated the maximal hypervolume according to the ground truth of all architectures.

Metric. We used the *log hypervolume* difference, the same as Daulton et al. (2020), as our evaluation criterion for NasBench201. This is because, in NasBench201, the performance difference between any two architectures may be small. Using log hypervolume allows us to visualize the sample efficiency of different algorithms more effectively. The metric is defined as:

$$HV_{\text{log\_diff}} := \log(HV_{\max} - HV_{\text{cur}}), \tag{5}$$

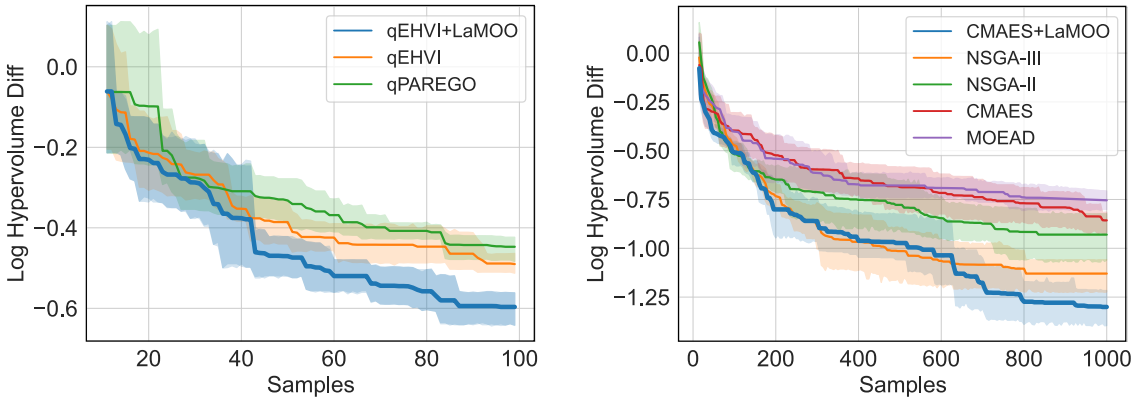
where HV_{cur} is the hypervolume of current samples obtained by the algorithm with a given budget. The smaller the log hypervolume difference, the better the performance.

Results. Figure 7 first compares the search performance of three NAS approaches when integrated with *LaMOO*. One-shot evaluation has the worst search performance due to the inaccurate accuracy estimation in either Bayesian search or random search. The few-shot version is better, and vanilla NAS, by training each architecture from scratch, performs best. Note that the inaccuracy of neural



(a) Bayesian optimization (b) Random search

Figure 7: One-shot, few-shot, and vanilla (i.e., evaluation with trained architectures) NAS comparison on NasBench201. We ran each algorithm seven times (the shaded areas are $\pm$ std of the mean).

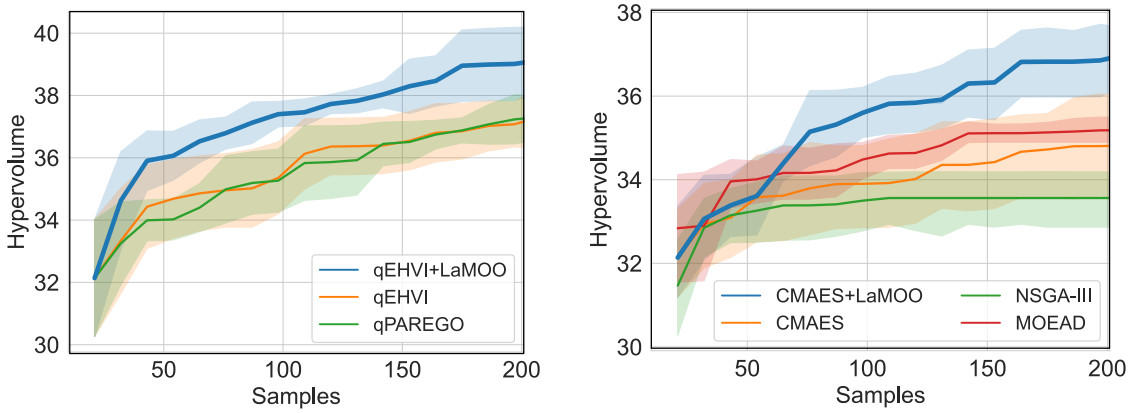


(a) Bayesian optimization (b) Evolutionary search

Figure 8: The search performance of *LaMOO* on NasBench201. We ran each algorithm seven times (the shaded area are $\pm$ std of the mean).

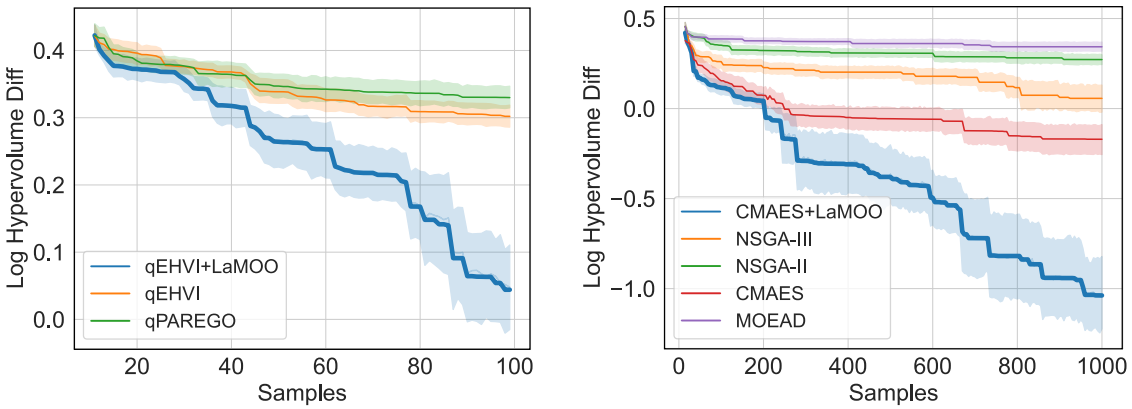
architecture performance estimation by one-shot or few-shot NAS may mislead *LaMOO*. Specifically, architectures that appear promising based on supernet estimations but actually have low accuracy may be considered good architectures for classification. This misidentification can lead *LaMOO* to focus subsequent sampling efforts in non-promising regions, thus degrading overall search performance as most generated samples may come from the non-promising regions and not effectively guide search directions or space partitioning. However, the exploration component within the MCTS has the potential to fix the search direction by preventing *LaMOO* from being confined to these less promising areas.

Next, we evaluate the efficacy of *LaMOO* as a meta-algorithm integrated with vanilla NAS. As shown in Figure 8, *LaMOO* with qEHVI outperforms all our BO baselines, and *LaMOO* with CMA-ES outperforms all our EA baselines, in terms of $HV_{\text{log\_diff}}$. Specifically, LaMOO+qEHVI



(a) Bayesian optimization (b) Evolutionary search

Figure 9: The search performance of *LaMOO* on NasBench301. We ran each algorithm seven times (the shaded areas are $\pm$ std of the mean).



(a) Bayesian optimization (b) Evolutionary search

Figure 10: The search performance of *LaMOO* on HW-NAS-BENCH. We ran each algorithm seven times (the shaded areas are $\pm$ std of the mean).

achieves 225% sample efficiency compared to other BO algorithms on Nasbench201. In addition, evolutionary algorithms can be trapped into local optima because they rely on mutation and crossover of previous samples to generate new ones. By using MCTS, LaMOO+CMA-ES can explore the search space between different iterations, greatly improving upon CMA-ES in NasBench201. In short, this result indicates that space partitioning, the core of *LaMOO*, leads to faster and better optimization in NAS-based problems.

6.1.2 EVALUATIONS USING NASBENCH301

Dataset overview. NasBench301 Zela et al. (2022) is a surrogate benchmark for the NASNet Zoph et al. (2018) search space, which contains more than 10^{21} architectures. Specifically, NasBench301

leverages a surrogate model to fit on a subset of architectures ($\sim$60k architectures), and predict the architecture's performance in the entire DARTS search space on the CIFAR10 image classification task. The surrogate model in NasBench301 demonstrates accurate regression results for evaluating the architectures in the large search space. NasBench301 provides the architecture accuracy from the surrogate model. Other basic metrics of architectures, such as #Params, #FLOPs, and inference time, can be easily measured in the evaluation process. For this multi-objective optimization, we maximize the inference accuracy and minimize #Params at the same time in the NasBench301 search space. For NASNet search space, it contains the operations of 3x3 max pool, 3x3, 5x5, depth-separable conv, and skip connection. The search target is to get the architectures for a reduction and a normal cell, and the number of nodes within a cell is 4. This formulates a search space of 3.5 x 10^{21} architectures Zela et al. (2022); Wang et al. (2019a). We use the same encoding method as the NASNet search space for the open-domain CIFAR10 problem and encode the architectures in the NASNet search space with vectors of 16 numbers. Specifically, the first 4 elements represent the operations in a normal cell, 5-8 elements are the concatenation type of the normal cell, 9-12 elements represent the operations in the reduction cell, and the last four elements are the concatenation type of reduction cell. Similar to NasBench201, we normalized #Params to range $[-1, 0]$ and accuracy to $[0, 1]$.

Metric. Since NasBench301 is a large-scale search space, the maximum hypervolume of this search space is unknown. Instead of using the hypervolume difference, we directly leverage the hypervolume value to demonstrate the performance of the search.

Results. As shown in Figure 9, *LaMOO* with qEHVI outperforms all our BO baselines, and *LaMOO* with CMA-ES outperforms all our EA baselines, in terms of hypervolume. In terms of Bayesian optimization-based algorithms, qEHVI+LaMOO has 200% search performance improvement compared to qEHVI and qPAREGO. For evolutionary-based algorithms, CMAES with *LaMOO* improves more than 250% sample efficiency compared to other baselines. It also largely outperforms other baselines after 200 search samples. This result indicates that *LaMOO* can effectively solve the high-dimensional NAS problems, 16 in the case of Nasbench301.

6.1.3 MANY-OBJECTIVE SEARCH ON HARDWARE-AWARE NASBENCH

Dataset overview. HW-NAS-Bench Li et al. (2021) extends the original NasBench201 dataset by providing additional information that facilitates the consideration of a broader spectrum of metrics (e.g., inference latency, parameter size, #FLOPs, etc.). This dataset enables us to evaluate the performance of multi-objective neural architecture search. We adopted the same architecture encoding strategy as NasBench201, as detailed in §6.1.1.

In our search process, we search for four distinct metrics: accuracy, number of parameters, #FLOPs, and inference latency on edge GPU. We selected these metrics because they are crucial indicators that model designers tend to prioritize Zoph et al. (2018); Real et al. (2019); He et al. (2016); Wang et al. (2022); Radosavovic et al. (2020a). Note that these metrics vary significantly in their range scales due to differing units of measurement. For instance, the highest number of FLOPs recorded is 220 million, whereas the longest inference latency is merely 0.024 seconds. Such disparities in scale could disproportionately bias the search toward metrics with larger numerical values. To mitigate this issue, we employ a min-max normalization technique that re-scales the values to a uniform range between 0 and 1, which allows us to leverage the complete architecture information available within the entire search space.

Metric. Similar to NasBench201, we employ the log hypervolume difference to assess the search performance of various algorithms. The specifics of this metric are detailed in §6.1.1.

Results. Figure 10 illustrates a comparison between *LaMOO* and other baselines. As depicted in Figure 10a, our *LaMOO*, when coupled with qEHVI, exhibits superior performance, significantly surpassing other Bayesian optimization baselines in terms of hypervolume. Figure 10b indicates that evolutionary-based search algorithms tend to reach a performance plateau when addressing many-objective NAS problems. However, when *LaMOO* is paired with CMA-ES, *LaMOO* helps it escape the initial region to focus on a smaller promising region by space partitioning. Figure 10b shows that CMA-ES+*LaMOO* achieves a search performance increase of over 250% compared to other evolutionary-based baselines. For example, CMA-ES+*LaMOO* achieves better hypervolume value with 400 samples than CMA-ES with 1000 samples. These results demonstrate the efficacy of *LaMOO* in many-objective neural architecture search tasks.

While *LaMOO* exhibits strong performance in many-objective NAS tasks, there is a potential limitation in the search process. As the number of objectives increases, even with the implementation of *leaf selection*, computing the hypervolume value for samples within each leaf node of the Monte Carlo Search Tree is challenging. As detailed in §4.2.1, the computational cost for hypervolume evaluation grows exponentially. This increase may slow down the search speed, particularly as the number of samples expands with the search progress.

6.2 Open-Domain NAS Tasks

To further demonstrate the search performance of *LaMOO* on open-domain NAS problems, we use *LaMOO* to search for architectures on CIFAR-10 using the NASNet search space, and on ImageNet using the EfficientNet search space. We also leverage *LaMOO* to search for architectures on object detection tasks and on Penn Treebank Language models.

6.2.1 SEARCHING ON CIFAR10 IMAGE CLASSIFICATION TASK

Search space overview. Our search space aligns with the NASNet search space Zoph et al. (2018). Specifically, it includes a total of eight searchable operations: 3x3 max pooling, 3x3 average pooling, 3x3, 5x5, and 7x7 depthwise convolutions, 3x3 and 5x5 dilated convolutions, and skip connection. The architecture is comprised of normal cells, which maintain the feature map's size, and reduction cells that both upscale the channels by a factor of two and down-sample the resolution of the feature map by two. Each cell incorporates 4 nodes connected by 8 different operations. There are a total of 10^{21} architectures in the search space. We use the same encoding strategy as previous work Wang et al. (2019a). Each sampled network is trained for 600 epochs, with a batch size of 128, using a momentum SGD optimizer initiated with a learning rate of 0.025, which is then subject to a cosine learning rate schedule throughout the training period. Weight decay is employed for regularization purposes.

Results. We apply our *LaMOO* with three mainstream NAS evaluation methods, i.e., vanilla NAS, one-shot NAS Guo et al. (2019), and few-shot NAS Zhao et al. (2021). Table 4 summarizes the SoTA results with DARTS and NASNet search space on CIFAR10, where the first group is models found with vanilla NAS and the second group with one-shot NAS. For *LaMOO*, we pick the best architectures from the Pareto frontier, where the accuracies are acquired by actual training for vanilla NAS and supernet(s) estimation for one-shot NAS. *LaMOO* with qEHVI finds architectures with similar accuracy to ones by vanilla NAS search, with fewer #Params at 1.62M, compared to at least

Table 4: Results on CIFAR-10 using the NasNet search space. The two optimization objectives we are searching for are #params and accuracy. Here, LaMOONet-P1 and LaMOONet-P2 represent architectures searched by LaMOO with smaller and larger parameter sizes, and one-shot LaMOONet and few-shot LaMOONet represent architectures discovered by LaMOO using one-shot and few-shot method. Baseline methods are sorted in descending order of test error.

Method	Evaluation Method	#Parameters	#Samples	Test Error(%)	GPU days
PNASNet-5 Liu et al. (2018)	Standard	3.2M	1160	3.41±0.09	225
NAO Luo et al. (2018)	Standard	3.2M	1000	3.14±0.09	225
NASNet-A Zoph et al. (2018)	Standard	3.3M	20000	2.65	2000
LEMONADE Elsken et al. (2018)	Standard	13.1M	-	2.58	90
AlphaX Wang et al. (2019b)	Standard	2.83M	1000	2.54±0.06	1000
AmoebaNet-B-small Real et al. (2019)	Standard	2.8M	27000	2.50±0.05	3150
LaMOONet-P1	Standard	**1.62M**	600	2.64±0.03	100
LaMOONet-P2	Standard	3.25M	600	**2.23**±0.06	100
BayeNAS Zhou et al. (2019)	One-shot	3.4M	N/A	2.81±0.04	0.2
DARTS Liu et al. (2019)	One-shot	3.3M	N/A	2.76±0.09	1.0
MergeNAS Wang et al. (2020d)	One-shot	2.9M	N/A	2.68±0.01	0.6
One-shot REA	One-shot	3.5M	N/A	2.68±0.03	0.75
CNAS Lim et al. (2020)	One-shot	3.7M	N/A	2.60±0.06	0.3
PC-DARTS Xu et al. (2020)	One-shot	3.6M	N/A	2.57±0.07	0.3
Fair-DARTS Chu et al. (2019)	One-shot	3.32M	N/A	2.54±0.05	3.0
ASNG-NAS Akimoto et al. (2019)	One-shot	3.32M	N/A	2.54±0.05	0.11
P-DARTS Chen et al. (2019)	One-shot	3.4M	N/A	2.50	0.3
One-shot LaNas Wang et al. (2019a)	One-shot	3.6M	N/A	2.24±0.02	3.0
One-shot LaMOONet	One-shot	**1.68M**	N/A	2.85±0.08	1.18
Few-shot LaMOONet	One-shot	**1.65M**	N/A	2.78±0.05	2.06

2.8M. In addition, in terms of the search cost measured in GPU days, *LaMOO* takes 30X fewer samples to find the LaMOONet-P2 architecture which achieve better accuracy (97.77% test accuracy) than AmoebaNet Real et al. (2019), the best performing SoTA architecture found with vanilla NAS. The one-shot and few-shot LaMOONet also demonstrate strong results in terms of both #Params and accuracy compared to their counterparts. The performance gap between the one-shot (second group) and vanilla NAS (first group) methods is because of supernet's poor accuracy prediction Guo et al. (2019); Yu et al. (2019b). Few-shot NAS Zhao et al. (2021) narrows this performance gap by using multiple supernets, with a slightly increased search cost from 1.18 to 2.06 GPU days.

6.2.2 SEARCHING ON IMAGENET IMAGE CLASSIFICATION TASK

The ImageNet search space comes from EffcientNet Tan and Le (2019a). The depth of an Inverted Residual Block (IRB) can be 2, 3, or 4, along with 3 types of connection patterns; and the expansion ratio within an IRB can be 3, 5, 6, or 7. The kernel size is chosen from 3, 5, 7. Therefore, the total possible architectures are $((3*3*4)^2 + (3*3*4)^3 + (3*3*4)^4)^5 \approx 10^{31}$. The details of the connection search space can be found in Appendix A. For this task, prior work either developed a model on ImageNet that prioritizes accuracy and the number of #FLOPs Tan and Le (2019a); Howard et al. (2017); Zhao et al. (2021); Wang et al. (2019a,b); Zoph et al. (2018) or focused on accuracy and inference latency Wang et al. (2022); Cai et al. (2020). Consequently, we have conducted two NAS experiments on ImageNet: the first targets accuracy and the number of #FLOPs, and the second aims for accuracy while considering TensorRT latency with FP16 on an NVIDIA GV100. For the

Table 5: Results on ImageNet using the EfficientNet search space. The two optimization objectives we are searching for are #FLOPs and accuracy. Here, LaMOONet-F1 and LaMOONet-F2 represent architectures searched by LaMOO with smaller and larger #FLOPs. We report the metric values where applicable based on reported values from original publications. Baseline methods are sorted in ascending order of top-1 accuracy.

Method	**#FLOPs**	**#Params**	**Top-1 Acc(%)**	**GPU days**
REGNETY-400MF Radosavovic et al. (2020b)	400M	4.3M	74.1	-
AutoSlim Yu and Huang (2019)	305M	5.7M	74.2	-
MnasNet-A1 Tan et al. (2019)	312M	3.9M	75.2	-
MobileNet-V3-large Howard et al. (2019)	219M	5.8M	75.2	-
FairDARTS Chu et al. (2019)	440M	4.3M	75.6	3.0
FBNetV2-F4 Wan et al. (2020)	238M	5.6M	76.0	8.3
BigNAS Yu et al. (2020)	242M	4.5M	76.5	-
OFA-small Wang et al. (2019b)	230M	5.4M	76.9	1.6
MixNet-M Tan and Le (2019b)	360M	5.0M	77.0	-
EfficientNet-B0 Tan and Le (2019a)	390M	5.3M	77.3	-
AtomNAS Mei et al. (2020)	363M	5.9M	77.6	-
LaMOONet-F0	**248M**	5.1M	**78.0**	1.5
ChamNet Dai et al. (2019)	553M	-	75.4	-
RegNet Radosavovic et al. (2020a)	600M	6.1M	75.5	-
REGNETY-800MF Radosavovic et al. (2020b)	800M	6.3M	76.3	-
MixNet-L Tan and Le (2019b)	565M	7.3M	78.9	-
FBNetV3 Dai et al. (2020)	544M	-	79.5	-
OFA-large Cai et al. (2020)	595M	9.1M	80.0	1.6
EfficientNet-B2 Tan and Le (2019a)	1000M	6.1M	80.3	-
NSGANetV2-xl Lu et al. (2020)	593M	8.7M	80.4	1
LaMOONet-F1	**522M**	7.8M	**80.4**	1.5

TensorRT latency setup, we fixed the TensorRT workspace at 10GB for all runs and benchmarked the latency using a batch size of 1 with explicit shape configuration. We report the average latency derived from 1000 runs. Recall that we have demonstrated that one-shot LaMOONet and few-shot LaMOONet achieve similar performance in Table 4; therefore, we will focus on evaluating the search performance of *LaMOO* with the one-shot pipeline.

ImageNet training setup. For each architecture in the Pareto frontier, we train it using 8 Tesla V100 GPUs with images of a 224x224 resolution in (accuracy, #FLOPs) two-objective search. For the (accuracy, latency) two-objective search, we set the image resolution of searched architectures as 320x320. We use the standard SGD optimizer with Nesterov momentum 0.9 and set the weight decay to be 3×10^{-5}. Each architecture is trained for a total of 450 epochs, with the first 10 epochs as the warm-up period. During the warm-up epochs, we use a constant learning rate of 0.01. The remaining epochs are trained with an initial learning rate of 0.1, a cosine learning rate decay schedule Loshchilov and Hutter (2016), and a batch size of 1024 (i.e., 128 images per GPU). The model parameters are subject to a decay factor of 0.9997 to further improve the training performance of our models.

Results. Table 5 compares our *LaMOO* with others SoTA baselines with different #FLOPs scales. The results demonstrate that the searched architectures by *LaMOO* greatly outperform other baselines in terms of both #FLOPs and accuracy. We group state-of-the-art models by their #FLOPs. The models in the first group have #FLOPs less than 500M while the models in the second group have #FLOPs more than 500M. We pick our models, labeled LaMOONet-F0 and LaMOONet-F1, from

Table 6: Results on ImageNet using the EfficientNet search space. The two optimization objectives we are searching for are TensorRT Latency with FP16 in NVIDIA GV100 and accuracy. Here, LaMOONet-G0 and LaMOONet-G1 represent architectures searched by LaMOO with smaller and larger inference latency. Baseline methods are sorted in ascending order of top-1 accuracy.

Method	**Top-1 Acc(%)**	**TensorRT Latency FP16 GV100 (ms)**	**LaMOONet Speedup↑**	**LaMOONet Acc Improvement↑**
RegNet-X Radosavovic et al. (2020a)	77.0	2.06	3.6x	2.2
EfficientNet-B0 Tan and Le (2019a)	77.1	1.18	2.1x	2.1
FBNetV2-L1 Wan et al. (2020)	77.2	1.13	2.0x	2.0
EfficientNetX-B0-GPU Tan and Le (2019a)	77.3	1.05	1.8x	1.9
GPUNet-0 Wang et al. (2022)	78.9	0.62	1.1x	0.3
LaMOONet-G0	**79.2**	**0.57**	-	-
FBNetV3-B Dai et al. (2020)	79.8	1.55	2.2x	0.8
EfficientNetX-B2-GPU Tan and Le (2019a)	80.0	1.61	2.2x	0.6
RegNet-X Radosavovic et al. (2020a)	80.0	3.9	5.4x	0.6
EfficientNet-B2 Tan and Le (2019a)	80.3	1.86	2.6x	0.3
ResNet-50 He et al. (2016)	80.3	1.1	1.5x	0.3
GPUNet-1 Wang et al. (2022)	80.5	0.82	1.1x	0.1
LaMOONet-G1	**80.6**	**0.72**	-	-

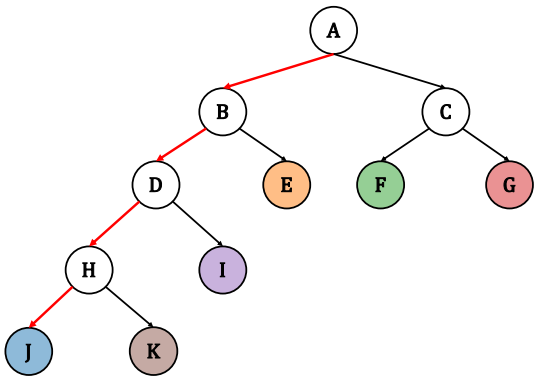
Table 7: Results on the test-dev set of MS COCO with different decoder, backbone, and channels. R-50 represent ResNet50 architecture He et al. (2016). Note that NAS-FCOS is the upgraded version of FCOS but we did not implement this on our backbone model. All networks have the same input image resolution.

Decoder	**Backbone**	**#Channels**	**#FLOPs(G)**	**AP**
FPN-FCOS Tian et al. (2019)	R-50 He et al. (2016)	256	169.9	37.4
NAS-FCOS Wang et al. (2020c)	MobileNetV2 Sandler et al. (2018)	128	39.3	32.0
NAS-FCOS Wang et al. (2020c)	MobileNetV2 Sandler et al. (2018)	256	121.8	34.7
FPN-FCOS Tian et al. (2019)	MobileNetV2 Sandler et al. (2018)	256	105.4	31.2
FPN-FCOS Tian et al. (2019)	**LaMOONet-D0**	128	**35.2**	**36.5**
FPN-FCOS Tian et al. (2019)	**LaMOONet-D1**	256	**109.5**	**37.6**

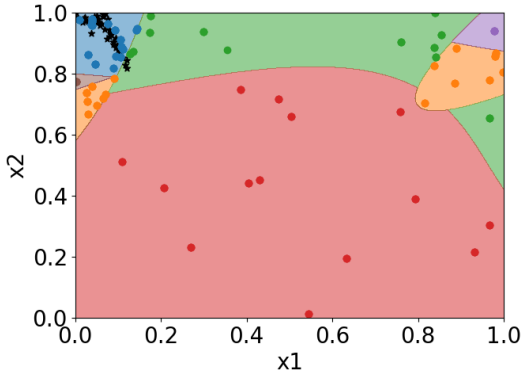
the Pareto frontier based on the two objectives, accuracy and #FLOPs. We see that LaMOONet-F1 has the highest accuracy, 0.1 higher than EfficientNet-B2, with only 52.2% #FLOPs. Similarly, LaMOONet-F0 also has the highest accuracy in its group, with the lowest #FLOPs. Table 6 compares our models found by *LaMOO* with other SoTA baselines with different inference latencies. The results show that the architectures found by *LaMOO* significantly surpass all baselines, delivering higher accuracy with reduced TensorRT inference time. Our LaMOONet-G0 achieves a 1.1X speedup over the SoTA architecture GPUNet-0 Wang et al. (2022) while having a 0.3% higher accuracy. LaMOONet-G1 has the highest top-1 accuracy of 80.6% while incurring the lowest TensorRT latency with FP16 on an NVIDIA GV100.

6.2.3 SEARCHING ON PENN TREEBANK

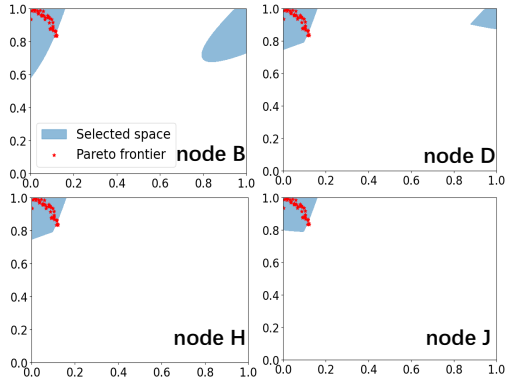
We evaluate *LaMOO* on Penn Treebank (PTB), a widely-studied benchmark for language models. We used the same search space and training setup as the original DARTS to search RNN on PTB.



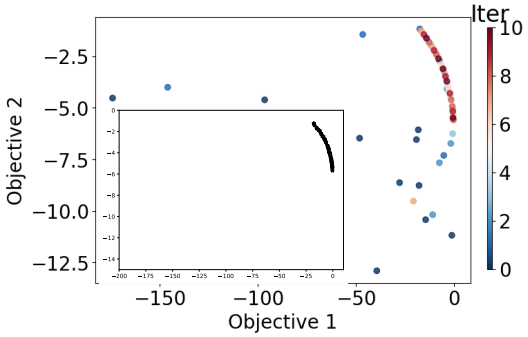
(a) The structure of Monte-Carlo Tree at the final search iteration.



(b) The samples and SVM split region of leaf node of the tree in the search space.



(c) The selected region in selected path of the tree.



(d) The samples in the objective space at different search iterations.

Figure 11: Visualization of selected region at different search iterations and nodes. (a) The Monte-Carlo tree with colored leaves. The selected path is marked in red. (b) Visualization of the regions $(\Omega_J, \Omega_K, \Omega_I, \Omega_E, \Omega_F, \Omega_G)$ that are consistent with leaves in (a) in the search space. Black dots represent the Pareto frontier estimated by 10^6 random samples. (c) Visualization of the selected path at final iteration. (d) Visualization of samples during search; bottom left is the Pareto frontier estimated from one million samples.

Here we search for two objectives, i.e., perplexity and #Params. By using *LaMOO*, we achieved the state-of-the-art test Perplexity of 54.56 with only 22M #Params. In comparison, the one-shot version DARTS algorithm found an architecture with worse performance (55.7 test Perplexity) and more #Params (23M); the few-shot version DARTS Zhao et al. (2021) found an architecture with slightly better performance (54.89 test perplexity) but requires 23M #Params.

6.2.4 SEARCHING ON MS COCO OBJECT DETECTION TASK

We leverage *LaMOO* searching for the efficient backbone with FPN-FCOS Tian et al. (2019) decoder (neck) in object detection. We also compared our searched architectures with the more effective FCOS-based decoder called NAS-FCOS Wang et al. (2020c), which leverages NAS for searching in promising FPN-FCOS architectures and has shown better results compared to FPN-FCOS. We use our

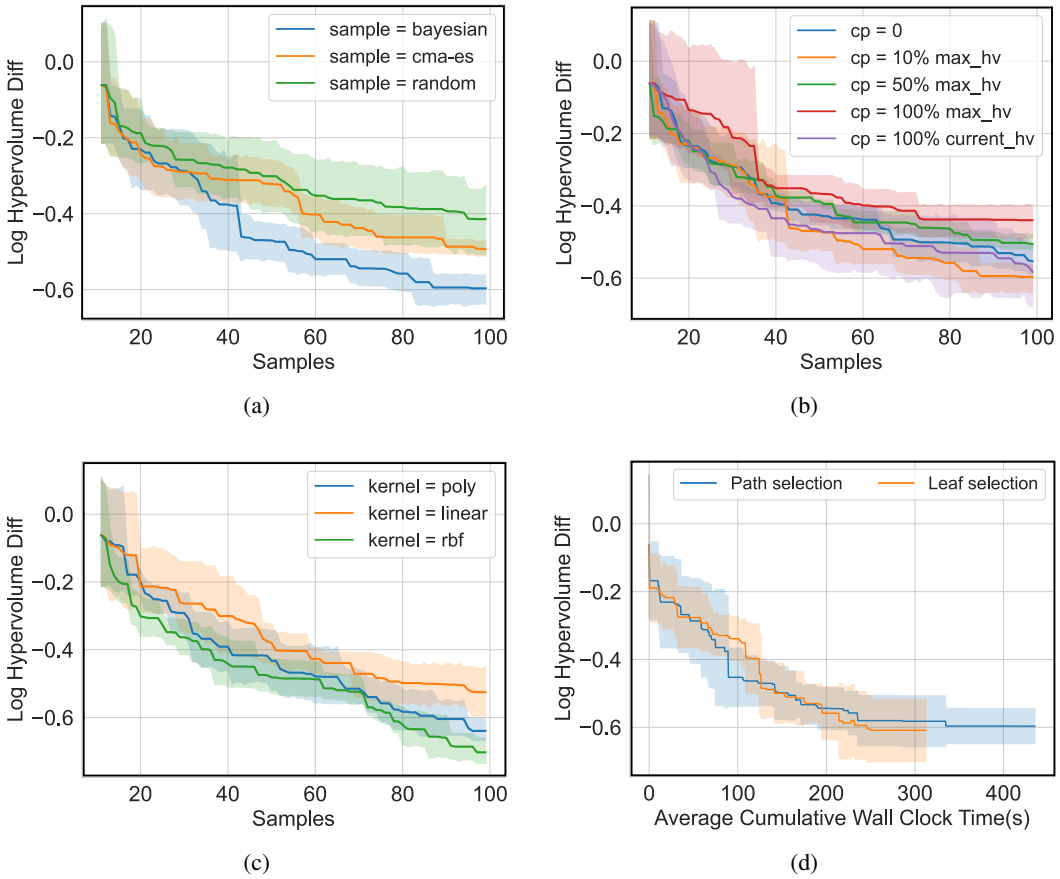


Figure 12: Ablation studies on hyperparameters and sampling methods in *LaMOO* on NasBench201. (a) Sampling with different methods. (b) Sampling with different C_p. (c) Partitioning with different SVM kernels. (d) Comparison of leaf selection and path selection methods in MCTS.

searched efficient backbones and compare the performance to the lightweight backbone MobileNetV2 and a more powerful but non-efficient based backbone ResNet50 (R-50). We implement both 128 and 256 channels of the decoder. The results on the MS COCO test set are shown in Table 7.

Object detection training setup. We use 4 Telsa V100 GPUs for training our models. We use the standard SGD optimizer with an initial learning rate of 0.005 and a norm gradient clip at 35. We train each model for 24 epochs and use the first 500 iterations as the warm-up phase. During the warm-up iterations, the learning rate starts at 0.002 and increases by 0.00033 every 50 iterations until it reaches 0.005. After the warm-up phase, we divide the learning rate by 10 at the 10^{th} and 22^{nd} epochs (i.e., to 5×10^{-4} and 5×10^{-5} respectively). We resize each image to 1333 by 800 and implement a random flip with a 0.5 flip ratio. Further, we apply the *center-ness* and *center* sampling techniques on our training pipeline, based on prior work Tian et al. (2019); Wang et al. (2020c) to further improve the results.

Results. Our searched backbone with 256 channels outperforms the ResNet50 with the same channel numbers and decoder by 0.2 AP (average precision) but with 60.4G fewer #FLOPs. In particular, compared to the mobile-based backbone, which shares the same design style with us, by

using FPN-FCOS with 128 channels, LaMOONet-D0 only requires 35.2G #FLOPs but achieves a promising AP with this scale of #FLOPs. These results greatly surpass the MobileNetV2 with 256 channels and a better decoder NAS-FCOS. In summary, we show that searching with *LaMOO* leads to architectures of better performance, measured in average precision, with low #FLOPs. In other words, *LaMOO* searched models have the potential to run on resource-constrained devices and achieve good detection accuracy for the object detection task.

6.3 Ablation Studies

6.3.1 VISUALIZATION OF LAMOO

To gain insights into the operation of *LaMOO*, we present a visualization of its optimization process for the Branin-Currin problem, a two-dimensional input problem, for ease of visualization. First, the Pareto optimal set Ω_P is estimated from 10^6 random samples (marked as black stars), as shown in both search and objective space (Figure. 11b and bottom left of Figure 11d. Over several iterations, *LaMOO* progressively prunes away unpromising regions so that the remaining regions approach Ω_P. Figure 11c shows that the selected nodes consist of promising regions. Figure 11a shows the final tree structure. The color of each leaf node corresponds to a region in the search space depicted in Figure 11b. The selected region is recursively bounded by SVM classifiers corresponding to nodes on the selected path (red arrows in Figure 11a). In the most promising region, Ω_J, we could implement any of the NAS and black-box optimization algorithms, such as reinforcement learning, evolutionary algorithms, and Bayesian optimization, to generate new samples. For a given optimization algorithm, we utilize SVM classifiers to define boundaries for the optimization algorithm, therefore confining the generation of new samples within Ω_J. This approach effectively enhances sample efficiency by focusing efforts on the most promising areas. §4.2.2 provides details of integration *LaMOO* with different optimization algorithms.

6.3.2 ABLATION OF DESIGN CHOICES

We assess the impact of various hyperparameters and sampling methods on the performance of *LaMOO*. This study is conducted using the NasBench201 dataset as outlined below.

Sampling methods. *LaMOO* can be combined with different sampling methods, including Bayesian Optimization (such as qEHVI) and evolutionary algorithms (such as CMA-ES). Figure 12a indicates that qEHVI substantially enhances performance when compared to random sampling, whereas CMA-ES yields only a slight improvement. These results are in line with our previous discovery that, in the context of MOO, BO generally outperforms EA in terms of search efficiency.

The exploration factor C_p controls the balance of exploration and exploitation. A larger C_p guides LaMOO to visit the sub-optimal regions more often. Based on the results in Figure 12b, greedy search ($C_p = 0$) leads to worse performance compared to a proper C_p value (i.e. 10% of maximum hypervolume), which justifies our usage of MCTS. On the other hand, over-exploration can also yield even worse results than a greedy search. Therefore, a rule of thumb is to set the C_p to be roughly 10% of the maximum hypervolume $HV_{\max}$. When $HV_{\max}$ is unknown, C_p can be dynamically set to 10% of the hypervolume of current samples in each search iteration. The final performances by using 10% HVmax and 10% current hypervolume are similar.

SVM kernels. As shown in Figure 12c, we find that the RBF kernel performs the best, in agreement with (Wang et al., 2020b). Thanks to the non-linearity of the polynomial and RBF kernels, their region partitions perform better compared to the linear one.

MCTS node selection. We compare the leaf selection and path selection methods in MCTS. Figure 12d shows that leaf selection significantly saves search time while achieving similar search performance.

7. Conclusion and Future Work

This work applies a novel multi-objective optimizer called *LaMOO* to the domain of neural architecture search. We demonstrate that *LaMOO* can be seamlessly integrated with three prevalent NAS evaluation methods, i.e., one-shot, few-shot, and performance predictor-based NAS. Through a series of comprehensive experiments, we highlight the superior sample efficiency of *LaMOO* compared to existing methodologies for various NAS tasks, including open-source NAS datasets and open-domain NAS tasks. For example, *LaMOO* has found state-of-the-art models on ImageNet with an impressive top-1 accuracy of 80.4% at 522M #FLOPS and 78.0% top-1 accuracy at 248 M #FLOPS. Additionally, on CIFAR10, *LaMOO* successfully searched an architecture that delivers 97.36% top-1 accuracy with only 1.62M #parameters. These compelling results collectively highlight the efficacy and potential of *LaMOO* as a significant tool in multi-objective neural architecture search.

Despite the great success we have demonstrated with *LaMOO* in NAS, there are still many fruitful directions to pursue. For example, in this work, we utilize an SVM classifier to partition the search space. Beyond SVM, numerous other classification models, such as deep neural networks, can be integrated into *LaMOO* to potentially further enhance the quality of space partitioning. Moreover, as a multi-objective meta-optimizer, *LaMOO* can be extended to various research domains beyond NAS, including molecule discovery Zhao et al. (2022), hyperparameter tuning Van Aken et al. (2017); Zhang et al. (2021), optimization of large language models Zhang et al. (2024), and other real-world applications.

Acknowledgments

We thank the anonymous reviewers for their constructive reviews. This work was supported in part by NSF Grants #2105564 and #2236987, a VMware grant, the Worcester Polytechnic Institute's Computer Science Department. Most results presented in this paper were obtained using CloudBank Boerner et al. (2023), which is supported by the National Science Foundation under award #1925001. This work also used Expanse at San Diego Supercomputer Center through allocation CIS230364 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

Appendix A. Redesign Connection Pattern in Inverted Residual Block for EfficientNet Search Space

A.1 Connection search space in Inverted Residual Block

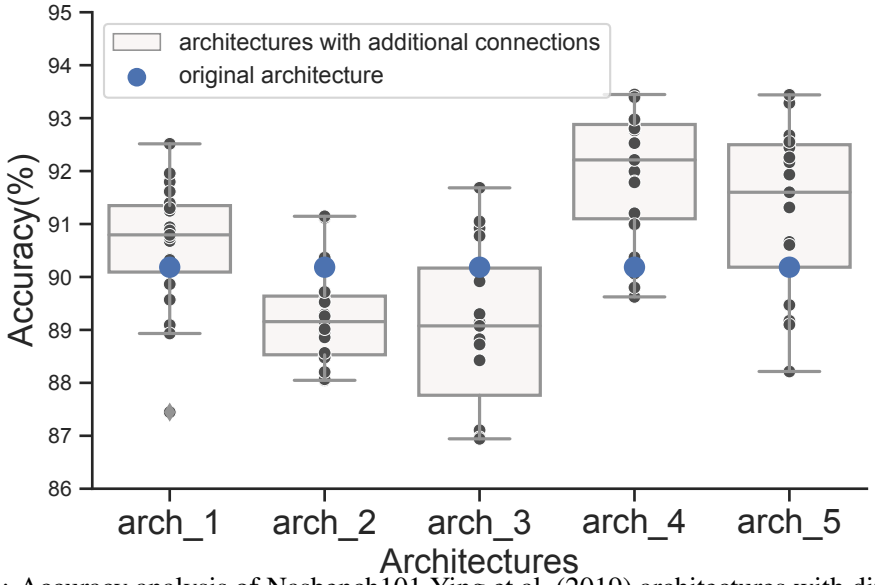
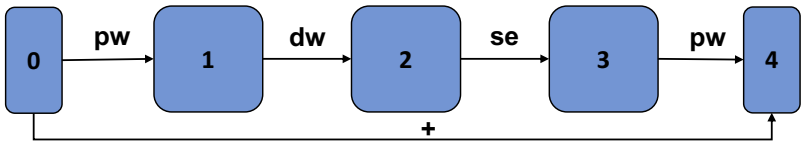


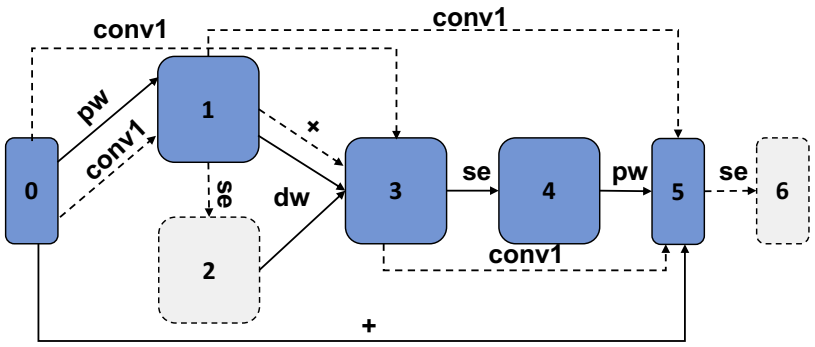
Figure 13: Accuracy analysis of Nasbench101 Ying et al. (2019) architectures with different connection patterns. We add different connections into the five sampled architectures (blue circle) and plot the corresponding accuracy distributions.

Existing work on efficient neural networks usually focuses on designing or modifying operation types of the IRB but often overlook another important design factor—the connection pattern Sandler et al. (2018); Howard et al. (2019); Wu et al. (2019); Wan et al. (2020); Cai et al. (2020, 2019). As demonstrated by prior works Liu et al. (2019); Wang et al. (2019b); Ying et al. (2019); Dong and Yang (2020); Zoph et al. (2018) and also observed by our own analysis (see Figure 13), the connection pattern of a neural architecture plays a crucial role in determining the model accuracy.

We randomly sample five architectures from the Nasbench101 dataset Ying et al. (2019), and use them as the basis for analyzing the impact of connections on model accuracy. Specifically, we add connections to the sampled architectures and plot the corresponding accuracy in Figure 13. We observe a wide range of accuracy distribution (e.g., can be up to tens percent difference) when architectures only differ in connections.



(a) The structure of inverted residual block(IRB)



(b) IRB with different connection patterns

Figure 14: (a): The architecture of inverted residual block(IRB), the width of the node indicates the channel numbers. (b): Our revised search space for the inverted residual block. The solid line represents the operations in original inverted residual block and the dash line represents the potential connections that can be searched in our new inverted residual block. The gray node denotes the potential node if corresponding SE operation is used. (PW: the point-wise convolution; DW: Depth-wise convolution; SE: squeeze and excite module.)

Table 8: FLOPs analysis for the IRB shown in Figure 14(b).

Operations between Node i and j(N_{ij})	**#FLOPs**
point-wise-conv1(N_{01})	$h_{irb} * w_{irb} * c_m * c_{io}$
depth-wise-convk(N_{13})	$h_{irb} * w_{irb} * c_m * k^2$
point-wise-conv1(N_{45})	$h_{irb} * w_{irb} * c_m * c_{io}$
grouped-conv1(N_{01})	$h_{irb} * w_{irb} * c_m$
grouped-conv1(N_{03})	$h_{irb} * w_{irb} * c_m$
grouped-conv1(N_{15})	$h_{irb} * w_{irb} * c_m$
grouped-conv1(N_{35})	$h_{irb} * w_{irb} * c_m$
se-module Hu et al. (2017)(N_{12})	$h_{irb} * w_{irb} * c_m * 2r$
se-module Hu et al. (2017)(N_{34})	$h_{irb} * w_{irb} * c_m * 2r$
se-module Hu et al. (2017)(N_{56})	$h_{irb} * w_{irb} * c_{io} * 2r$

To take the connection pattern into consideration, in this work, we redesign the connection space of the original inverted residual block (IRB) and design/add more connections, as shown in Figure 14. Rather than only linearly connecting the input and output layer with a shortcut as in the original

IRB(shown in Figure 14(a)), we consider all possible connections inside of the IRB to form our search space(shown in Figure 14(b)).

In the new IRB, all nodes have new connections (dashed lines). We use a 1x1 convolution operation to connect two nodes with mismatched channel numbers and enable SE module to integrate different locations in IRB. We do this because we would let attention mechanism by SE module to be impacted on all possible connections in the IRB.

In this paper, we searched the architecture in EfficientNet search space with this redesigned connection-searchable Inverted Residual Block using our *LaMOO*.

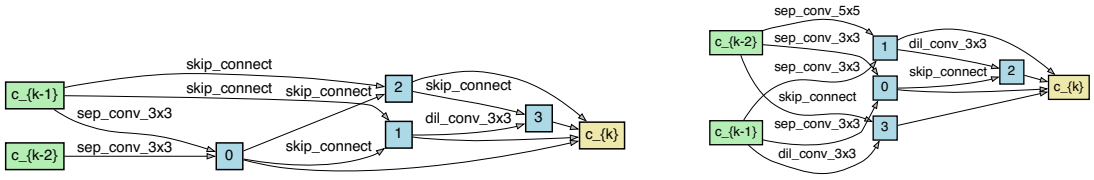
A.2 Connection search space in Inverted Residual Block

Note that one of the key performance goals of using IRB is to achieve low computation cost. In this section, we analyze the computation cost of our modified IRB (i.e,. cost of the added connections) and compare to that of the original IRB. Our analysis shows that adding connections only increases the computation cost slightly.

Cost of our new inverted residual block. In our search space of the connected IRB, we are only required to use 1x1 convolution to connect the mismatched channel dimensions four times as shown in Figure 14(b). These connections correspond to edges (N_0, N_1), (N_0, N_3), (N_1, N_5), (N_3, N_5), where the subscripts correspond to the node indexes. The total cost of these 1x1 convolution operation is $4 * (h_{irb} * w_{irb} * c_m * c_{io})$, which is not a negligible overhead. To reduce the cost caused by 1x1 convolution, we leverage the grouped convolution Zhang et al. (2018) in place of the standard convolution.

Specifically, we divide the output channel into number of input channel groups by first applying convolution operation in each corresponding input channel and then concatenating all the channels together. Therefore, the computational cost of these connections can be reduced to $4*(h_{irb}*w_{irb}*c_m)$ FLOPs if c_m is multiple of c_{io}. The new squeeze and excite modules placed after first and last PW layer take $2 * h_{irb} * w_{irb} * c_m * r$ and $2 * h_{irb} * w_{irb} * c_{io} * r$ respectively. Note that in IRB, c_{io} can be multiple times smaller than c_m. Therefore, the computational overhead associated with the added connections is bounded by $h_{irb} * w_{irb} * c_m * (k^2 + 2c_{io} + 6r + 4))$. As c_{io} grows faster than both r and k, the overhead ratio $4(r + 1)/(k^2 + 2 * (c_{io} + r))$ between our redesigned IRB and the original IRB is small.

Appendix B. Details of Searched Architectures



(a) Normal Cell (b) Reduction Cell

Figure 15: Founded cell structure on Cifar10.

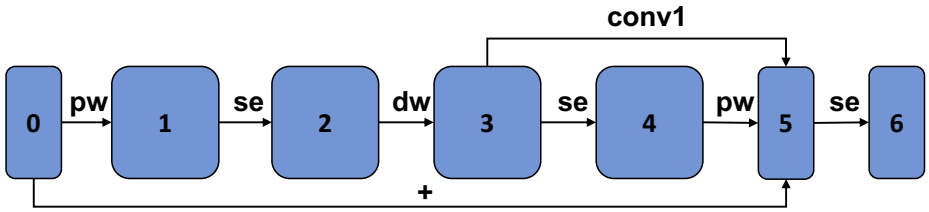


Figure 16: Searched connection and SE pattern in inverted residual block

Table 9: Searched ImageNet model with 248M #FLOPs.

Group	Block	kernel	Stride	Channel	Expand Ratio
0	Conv	3	2	24	-
	IRB	3	1	24	3
1	IRB	3	2	36	3
	IRB	3	1	36	3
	IRB	3	1	36	3
2	IRB	5	2	36	5
	IRB	5	1	36	5
	IRB	5	1	36	5
3	IRB	5	2	96	5
	IRB	5	1	96	3
	IRB	5	1	96	3
4	IRB	5	2	160	3
	IRB	5	1	160	6
	IRB	3	1	160	6
	IRB	3	1	160	7
5	Conv	3	1	960	-
	Conv	1	1	1280	-
	FC	1	1	1000	-

The best normal and reduction cell found by *LaMOO* is visualized in Figure 15. Refer to Liu et al. (2019); Zoph et al. (2018) for details about how to build a neural network with the searched cell. Table 9 demonstrates the founded architecture by *LaMOO* on the ImageNet dataset. Figure 16 is our searched connection pattern inside of IRB.

References

Nvidia's drive orin. https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/#:~:text=The%20NVIDIA%20DRIVE%20Orin%20SoC,digital%20clusters%2C%20and%20AI%20cockpits..

Youhei Akimoto, Shinichi Shirakawa, Nozomu Yoshinari, Kento Uchida, Shota Saito, and Kouhei Nishida. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 171–180, 2019.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Sanghamitra Bandyopadhyay, Sankar K Pal, and B Aruna. Multiobjective gas, quantitative indices, and pattern classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2088–2099, 2004.

Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Max-value entropy search for multi-objective bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F.d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/82edc5c9e21035674d481640448049f3-Paper.pdf.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning*, 10–15 Jul 2018.

Nicola Beume and Günter Rudolph. Faster s-metric calculation by considering dominated hypervolume as klee's measure problem, 2006.

Timothy J. Boerner, Stephen Deems, Thomas R. Furlani, Shelley L. Knuth, and John Towns. Access: Advancing innovation: Nsf's advanced cyberinfrastructure coordination ecosystem: Services & support. In *Practice and Experience in Advanced Research Computing*, PEARC '23, page 173–176, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399852. doi: 10.1145/3569951.3597559. URL https://doi.org/10.1145/3569951.3597559.

Peter AN Bosman and Dirk Thierens. The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE transactions on evolutionary computation*, 7(2):174–188, 2003.

Lucian Buşoniu, Alexander Daniels, Rémi Munos, and Robert Babuška. Optimistic planning for continuous-action deterministic systems. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 69–76. IEEE, 2013.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/pdf/1812.00332.pdf.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

Tianyi Chen, Luming Liang, Tianyu Ding, and Ilya Zharkov. Towards automatic neural architecture search within general super-networks. *arXiv preprint arXiv:2305.18030*, 2023.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive DARTS: bridging the optimization gap for NAS in the wild. *CoRR*, abs/1912.10952, 2019. URL http://arxiv.org/abs/1912.10952.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019. URL http://arxiv.org/abs/1911.12126.

Yuanzheng Ci, Chen Lin, Ming Sun, Boyu Chen, Hongwen Zhang, and Wanli Ouyang. Evolving search space for neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6659–6669, 2021.

X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha. Chamnet: Towards efficient network design through platform-aware model adaptation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11390–11399, 2019. doi: 10.1109/CVPR.2019.01166.

Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using neural acquisition function, 2020.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *arXiv preprint arXiv:2006.05078*, 2020.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014. doi: 10.1109/TEVC.2013.2281535.

Guoqiang Deng, Zhangcan Huang, and Min Tang. Research in the performance assessment of multi-objective optimization evolutionary algorithms. In *2007 International Conference on Communications, Circuits and Systems*, pages 915–918. IEEE, 2007.

Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 517–531, 2018.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.

Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: learning scalable feature pyramid architecture for object detection. *CoRR*, abs/1904.07392, 2019. URL http://arxiv.org/abs/1904.07392.

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. *CoRR*, abs/1904.00420, 2019. URL http://arxiv.org/abs/1904.00420.

Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, February 2019. URL https://doi.org/10.5281/zenodo.2559634.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Xin He, Jiangchao Yao, Yuxin Wang, Zhenheng Tang, Ka Chu Cheung, Simon See, Bo Han, and Xiaowen Chu. Nas-lid: Efficient neural architecture search with local intrinsic dimension. *arXiv preprint arXiv:2211.12759*, 2022.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. URL http://arxiv.org/abs/1905.02244.

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. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. URL http://arxiv.org/abs/1709.01507.

Shoukang Hu, Ruochen Wang, Lanqing HONG, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. Generalizing few-shot NAS with gradient matching. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=_jMtny3sMKU.

Christian Igel, Nikolaus Hansen, and Stefan Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, 15(1):1–28, 2007. doi: 10.1162/evco.2007.15.1.1.

Christian Igel, Thorsten Suttorp, and Nikolaus Hansen. Steady-state selection and efficient covariance matrix update in the multi-objective cma-es. In Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata, editors, *Evolutionary Multi-Criterion Optimization*, pages 171–185, Berlin, Heidelberg, 2007a. Springer Berlin Heidelberg.

Siwei Jiang, Yew-Soon Ong, Jie Zhang, and Liang Feng. Consistencies and contradictions of performance metrics in multiobjective optimization. *IEEE transactions on cybernetics*, 44(12): 2391–2404, 2014.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1): 50–66, 2006. doi: 10.1109/TEVC.2005.851274.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, October 1975. ISSN 0004-5411. doi: 10.1145/321906.321910. URL https://doi.org/10.1145/321906.321910.

Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan (Celine) Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=_0kaDkv3dVf.

Miqing Li, Shengxiang Yang, and Xiaohui Liu. Diversity comparison of pareto front approximations in many-objective optimization. *IEEE Transactions on Cybernetics*, 44(12):2568–2584, 2014.

Heechul Lim, Min-Soo Kim, and Jinjun Xiong. {CNAS}: Channel-level neural architecture search, 2020. URL https://openreview.net/forum?id=rklfIeSFwS.

Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *European Conference on Computer Vision(ECCV)*, 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations(ICLR)*, 2019.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL http://arxiv.org/abs/1608.03983.

Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, page 35–51, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58451-1. URL https://doi.org/10.1007/978-3-030-58452-8_3.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems 31*. 2018.

Renqian Luo, Tao Qin, and Enhong Chen. Balanced one-shot neural architecture optimization. abs/1909.10815, 2020. URL http://arxiv.org/abs/1909.10815.

Chris Mansley, Ari Weinstein, and Michael Littman. Sample-based planning for continuous action markov decision processes. In *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.

Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BylQSxHFwr.

Rémi Munos et al. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends® in Machine Learning*, 7(1):1–129, 2014.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning(ICML)*, 2018.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020a.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10428–10436, 2020b.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Association for the Advancement of Artificial Intelligence(AAAI)*, 2019.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Xiu Su, Shan You, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. K-shot nas: Learnable weight-sharing for nas with k-shot supernets. In *International Conference on Machine Learning*, pages 9880–9890. PMLR, 2021.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019a. URL http://arxiv.org/abs/1905.11946.

Mingxing Tan and Quoc V Le. Mixconv: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595*, 2019b.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Conference on Computer Vision and Pattern Recognition(CVPR)*, 2019.

Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proc. Int. Conf. Computer Vision (ICCV)*, 2019.

Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1009–1024, 2017.

David A Van Veldhuizen and Gary B Lamont. Evolutionary computation and convergence to a pareto front. In *Late breaking papers at the genetic programming 1998 conference*, pages 221–228. Citeseer, 1998.

Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning action space. *CoRR*, abs/1906.06832, 2019a. URL http://arxiv.org/abs/1906.06832.

Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *CoRR*, abs/1903.11059, 2019b. URL http://arxiv.org/abs/1903.11059.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using monte carlo tree search. *Advances in Neural Information Processing Systems*, 33:19511–19522, 2020a.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using monte carlo tree search. *arXiv preprint arXiv:2007.00708*, 2020b.

Linnan Wang, Chenhan D. Yu, Satish Salian, Slawomir Kierat, Szymon Migacz, and Alex Fit-Florea. Searching the deployable convolution neural networks for gpus. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12217–12226, 2022. URL https://api.semanticscholar.org/CorpusID:248496348.

Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020c.

Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3065–3072. ijcai.org, 2020d. doi: 10.24963/ijcai.2020/424. URL https://doi.org/10.24963/ijcai.2020/424.

Ari Weinstein and Michael L Littman. Bandit-based planning and learning in continuous-action markov decision processes. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

Bichen Wu. *Efficient deep neural networks*. University of California, Berkeley, 2019.

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. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Xin Xia, Xuefeng Xiao, Xing Wang, and Min Zheng. Progressive automatic design of search space for one-shot neural architecture search. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2455–2464, 2022.

Dongkuan Xu, Subhabrata Mukherjee, Xiaodong Liu, Debadeepta Dey, Wenhui Wang, Xiang Zhang, Ahmed Hassan Awadallah, and Jianfeng Gao. Few-shot task-agnostic neural architecture search for distilling large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=GdMqXQx5fFR.

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. URL https://openreview.net/forum?id=BJlS634tPr.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Jiahui Yu and Thomas S. Huang. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *CoRR*, abs/1903.11728, 2019. URL http://arxiv.org/abs/1903.11728.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020.

Kaicheng Yu, Rene Ranftl, and Mathieu Salzmann. How to train your super-net: An analysis of training heuristics in weight-sharing nas. 2019a.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019b.

Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=OnpFa95RVqs.

Kai Zhang, Lizhi Qing, Yangyang Kang, and Xiaozhong Liu. Personalized llm response generation with parameterized memory injection. *arXiv preprint arXiv:2404.03565*, 2024.

Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007. doi: 10.1109/TEVC.2007.892759.

X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. doi: 10.1109/CVPR.2018.00716.

Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2102–2114, 2021.

Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12707–12718. PMLR, 18–24 Jul 2021. URL http://proceedings.mlr.press/v139/zhao21d.html.

Yiyang Zhao, Linnan Wang, Kevin Yang, Tianjun Zhang, Tian Guo, and Yuandong Tian. Multi-objective optimization by learning space partition. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=FlwzVjfMryn.

Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. BayesNAS: A Bayesian approach for neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7603–7613. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/zhou19e.html.

Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc Le. Learning transferable architectures for scalable image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.