Neural Architecture Search as Multiobjective Optimization Benchmarks: Problem Formulation and Performance Assessment

Zhichao Lu, *Member, IEEE*, Ran Cheng, *Senior Member, IEEE*, Yaochu Jin, *Fellow, IEEE*, Kay Chen Tan, *Fellow, IEEE*, and Kalyanmoy Deb, *Fellow, IEEE* .

***Abstract*—The ongoing advancements in network architecture design have led to remarkable achievements in deep learning across various challenging computer vision tasks. Meanwhile, the development of neural architecture search (NAS) has provided promising approaches to automating the design of network architectures for lower prediction error. Recently, the emerging application scenarios of deep learning (e.g., autonomous driving) have raised higher demands for network architectures considering multiple design criteria: number of parameters/weights, number of floating-point operations, inference latency, among others. From an optimization point of view, the NAS tasks involving multiple design criteria are intrinsically multiobjective optimization problems; hence, it is reasonable to adopt evolutionary multiobjective optimization (EMO) algorithms for tackling them. Nonetheless, there is still a clear gap confining the related research along this pathway: on the one hand, there is a lack of a general problem formulation of NAS tasks from an optimization point of view; on the other hand, there are challenges in conducting benchmark assessments of EMO algorithms on NAS tasks. To bridge the gap: (i) we formulate NAS tasks into general multi-objective optimization problems and analyze the complex characteristics from an optimization point of view; (ii) we present an end-to-end pipeline, dubbed `EvoXBench`, to generate benchmark test problems for EMO algorithms to run efficiently – without the requirement of GPUs or Pytorch/Tensorflow; (iii) we instantiate two test suites comprehensively covering two datasets, seven search spaces, and three hardware devices, involving up to eight objectives. Based on the above, we validate the proposed test suites using six representative EMO algorithms and provide some empirical analyses. The code of `EvoXBench` is available at https://github.com/EMI-Group/EvoXBench.**

***Index Terms*—Evolutionary multi-objective optimization, neural architecture search, deep learning.**

Z. Lu and R. Cheng are with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. E-mail: {luzhichaocn, ranchengcn}@gmail.com (*Corresponding author: Ran Cheng*).

Y. Jin is with the Chair of Nature Inspired Computing and Engineering, Faculty of Technology, Bielefeld University, 33615 Bielefeld, Germany, and also with the Department of Computer Science, University of Surrey, Guildford, Surrey GU2 7XH, U.K. Email: yaochu.jin@uni-bielefeld.de.

K. C. Tan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR. E-mail: kctan@polyu.edu.hk

K. Deb is with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI 48824, USA. E-mail: kdeb@msu.edu

This work was supported by the National Natural Science Foundation of China (No. 62106097, 61906081), China Postdoctoral Science Foundation (No. 2021M691424), the Shenzhen Peacock Plan (No. KQTD2016112514355531), the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (No. 2017ZT07X386). Y. Jin is funded by an Alexander von Humboldt Professor for Artificial Intelligence endowed by the German Federal Ministry of Education and Research.

I. Introduction

With the development of deep neural networks (DNNs), deep learning has seen widespread growth in both research and applications, driven by improvements in computing power and a massive amount of data [1]–[3]. As the DNN models become more complex, it has come to a consensus that the manual process of designing DNN architectures is beyond human labor and thus unsustainable [4]–[6]. Meanwhile, the emergence of neural architecture search (NAS) has paved a promising path towards alleviating this unsustainable process by automating the pipeline of designing DNN architectures [7].

In the early days, NAS was mainly dedicated to obtaining DNN models with low prediction error – the most important target of most deep learning tasks. More recently, however, the emerging application scenarios of deep learning (e.g. autodriving) have raised higher demands for DNN architecture designs [8]. Consequently, apart from the prediction error, there are also other design criteria to consider, such as the number of parameters/weights and the number of floating-point operations (FLOPs). Particularly, when deploying DNN models to edge computing devices, the hardware-related performance is equally important, such as inference latency and energy consumption. Hence, from an optimization point of view, NAS tasks of today are intrinsically multiobjective optimization problems (MOPs) aiming to achieve optima of the multiple design criteria of the DNN architectures [9], [10].

Due to the black-box nature of NAS, the complex properties from the optimization point of view (e.g. discrete decision variables, multi-modal and noisy fitness landscape, expensive and many objectives, etc.) have posed great challenges to conventional optimization methods driven by rigorous mathematical assumptions. Accordingly, it is intuitive to resort to evolutionary multiobjective optimization (EMO) algorithms, which have well-established track records for handling complex MOPs [11]–[13]. Nonetheless, the progress of adopting EMO algorithms for NAS still lags well behind the progress of the general field of NAS research, due to the clear gap between the two fields caused by the following four main issues.

❶ **Computational Cost:** Many baseline NAS methods require enormous computational cost [14]–[16], which is unfriendly to the researchers in academia; particularly, EMO algorithms often require a large number of function evaluations to run, such that the issue of expensive computational cost will become especially sharp.

❷ **Algorithm Comparison:** Despite that the recent development has led to more efficient methods [17], [18], different NAS methods are not directly comparable to each other given the inherent differences in types of architectures (i.e., search spaces) and training procedures (i.e., fitness evaluations); hence, with the NAS methods in the current literature, it is very unlikely (if not impossible) to isolate the contribution of the optimization algorithm itself to the overall success of a NAS method [19].

❸ **Problem Formulation:** The research in EMO is often dedicated to solving MOPs with specific complex characteristics (e.g. multiple modalities, many objectives), but there is still no general formulation and analysis of NAS from the optimization point of view, thus having hindered EMO researchers to look deeper into the problems.

❹ **Running Environment:** Performing NAS usually requires expertise in deep learning – not only the experience in developing DNN models but also the knowledge in configuring the software/hardware environment; however, the conventional EMO researchers mainly focus on how to design efficient EMO algorithms, but not the environments to call/run the MOPs [20].

Among the four issues, issues ❶ and ❷ are not only for EMO but also for general NAS research. To address these two issues, the recent NAS literature has made some attempts to propose the so-called *tabular* NAS benchmarks [21]–[23]. The basic idea is to rely on an *exhaustive* evaluation of *all* attainable architectures for multiple repetitions, resulting in confined search spaces and toy-scale DNN architectures that are unrealistic for real-world deployments or downstream tasks. To this end, the concept of *surrogate* NAS benchmarks have been introduced [24]. It is expected that an accurate interpolation of the fitness landscape can be achieved on top of fix-budget training samples, thus extending NAS benchmarks to large search spaces with more than 10^{18} different DNN architectures. This recent progress in NAS benchmarks has led to a substantial reduction in the compute time required for NAS algorithm development and a significant improvement in the reproducibility of NAS research.

Despite the recent progress in addressing issues ❶ and ❷, the outcome architectures (from a NAS algorithm) still need to be re-trained from scratch as the optimal weights are either not provided by or not available in the existing NAS benchmarks [21], [24]. Given a set of Pareto-optimal architectures returned by an EMO algorithm, such a re-training process itself can render the entire approach computationally prohibitive. Nonetheless, there is still little effort dedicated to addressing issues ❸ and ❹.

Generally speaking, the challenges of issues ❸ and ❹ are two-fold: from the research perspective, existing NAS benchmarks are merely provided as datasets, but not constructed as benchmark test problems on the basis of general formulation from the EMO point of view; from an engineering perspective, existing NAS benchmarks still require installations/configurations of sophisticated deep learning software/hardware environments, e.g., GPU, TensorFlow, PyTorch, among others. Moreover, each of the existing NAS benchmarks merely covers one or two search spaces, which is far from sufficient for benchmark comparisons between EMO algorithms; however, if one hopes to test an algorithm over different benchmarks, he/she has to make repeated modifications to have the algorithm compatible to the complicated interfaces of each benchmark, thus making the development of NAS algorithms tedious and error-prone.

This paper is dedicated to bridging the gap between EMO and NAS by addressing the four issues above. In summary, the main contributions are:

– We present a general formulation of NAS problems from the multiobjective optimization perspective, involving three categories of optimization objectives: the prediction error objective, the complexity-related objectives, and the hardware-related objectives. On top of this formulation, we demonstrate a series of complex characteristics of NAS problems, including multiple modalities, many objectives, noisy objectives, degenerated Pareto fronts, among others.

– We provide a unified and comprehensive NAS benchmark, dubbed `EvoXBench`, comprising seven search spaces. Specifically, `EvoXBench` covers two types of architectures (convolutional neural networks and vision Transformers), two widely-studied datasets (CIFAR-10 and ImageNet), six types of hardware (GPU, mobile phone, FPGA, among others.), and up to six types of optimization objectives (prediction error, FLOPs, latency, among others.).

– We provide delicate engineering designs of `EvoXBench` to be user-friendly to EMO research. Specifically, `EvoXBench` can be *easily installed* without dependencies of deep learning software packages such as Pytorch or TensorFlow; and it can be *directly called* in any programming language such as Python, MATLAB, or Java; most importantly, running without GPU, `EvoXBench` provides *instant feedback* to MOEAs for real-time fitness evaluations.

– On the basis of our `EvoXBench`, we generate two representative benchmark test suites, i.e., the C-10/MOP and the IN-1K/MOP, involving MOPs with up to six objectives. Furthermore, we conduct a series of experiments to study the properties of the test problems using six representative MOEAs (NSGA-II, MOEA/D, IBEA, NSGA-III, HypE, and RVEA).

In the remainder of this paper, first, we provide necessary background information in Section II; second, we provide formal NAS problem formulation and analysis in Section III; third, we explain the design principles of our benchmark and test suite generation in Section IV and V, respectively; fourth, we provide empirical evaluations in Section VI; finally, conclusions and future studies are discussed in Section VII.

II. Background

In this section, we provide a brief overview of the related concepts. The main notations used in this paper are summarized in Table I.

A. Evolutionary Multiobjective Optimization

Generally, a multiobjective optimization problem (MOP), involving more than one conflicting objective to be optimized simultaneously, can be formulated as:

TABLE I: Summary of notations.

Category	Notation	Description
Data	C-10	CIFAR-10: 10-class image classification dataset [25]
	IN-1K	ImageNet 1K: 1,000-class image classification dataset [1]
	$\mathcal{D}_{trn}, \mathcal{D}_{vld}, \mathcal{D}_{tst}$	train, validation, and test splits of a dataset
Hardware	h / $\mathcal{H}$	a specific / set of hardware device
Decision vectors	$\boldsymbol{x} \in \Omega$	architecture decision variables and search space
	$\boldsymbol{w}(\boldsymbol{x})$	weights of an architecture
	D	number of decision variables
Objectives	f^e	prediction error on $\mathcal{D}_{vld}$
	$\boldsymbol{f}^c$	model complexity, i.e., # of parameters and FLOPs
	$\boldsymbol{f}^{\mathcal{H}}$	hardware efficiency, e.g., latency, energy consumption, etc.
	$\mathcal{L}_{trn}$	loss on $\mathcal{D}_{trn}$ for training $\boldsymbol{w}(\boldsymbol{x})$
	M	number of objectives
Others	N	population size
	MAE	mean absolute error
	τ	Kendall rank correlation coefficient

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{F}(\mathbf{x}) = \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_M(\boldsymbol{x})\big), \\ \text{s.t.} \quad & \boldsymbol{x} \in X, \quad \boldsymbol{F} \in Y, \end{aligned} \tag{1}$$

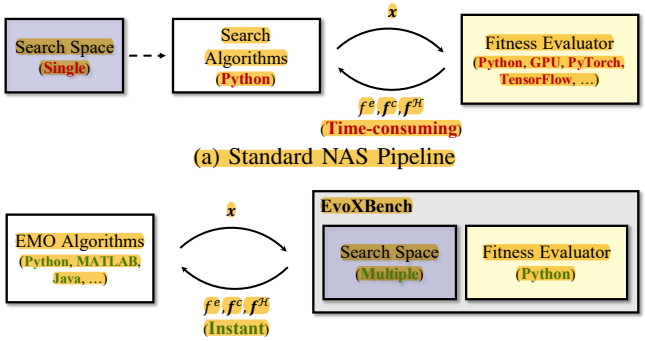
where $X \subset \mathbb{R}^D$ and $Y \subset \mathbb{R}^M$ are known as the *decision space* and the *objective space* respectively. Since $f_1(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x})$ are often conflicting with each other, there is no single solution that can achieve optima on all objectives simultaneously; instead, the optima to such an MOP is often a set of solutions trading-off between different objectives, known as the *Pareto optima*. Specifically, the images of the Pareto optima are known as the *Pareto set* (PS) and the *Pareto front* (PF) in the decision space and objective space respectively.

In practice, the target of solving an MOP is to approximate the PS/PS with a limited number of candidate solutions trading-off between different objectives. To this end, various EMO algorithms have been proposed during the past two decades. Generally, the EMO algorithms can be categorized into three categories: dominance-based ones [26]–[28], decomposition-based ones [12], [29], and performance-indicator-based ones [30], [31]. Readers are referred to supplementary materials §I-A for more details.

To assess the performance of various EMO algorithms in the literature, a number of challenging EMO benchmark test suites have also been designed to consider different complicated characteristics, e.g., the ZDT test suite [32], the DTLZ test suite [33], the WFG test suite [34], the MaF test suite [35], the LSMOP test suite [36], etc. Undoubtedly, the EMO benchmark test suites play an important role in pushing the boundaries of EMO research, having promoted the ongoing delicate designs of EMO algorithms to face the new challenges of emerging benchmark test suites.

B. Neural Architecture Search (NAS)

The goal of NAS is to automate the design of the architectures of DNN models by formulating it as an optimization problem [7]. As depicted in Fig. 1a, a standard NAS pipeline involves three main components: (i) a search space that predefines how a DNN architecture is represented; (ii) a search algorithm for generating suitable architectures according to pre-specified criteria; and (iii) an evaluator to estimate the fitness of an architecture [19]. In the following, we provide a brief review of the literature related to NAS search spaces



(a) Standard NAS Pipeline

(b) Our NAS Benchmark tailored for EMO

Fig. 1: Overall system-level comparison.

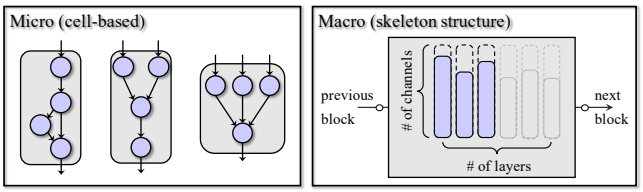


Fig. 2: Two types of widely used search spaces.

and refer readers to supplementary materials §I-B for NAS search algorithms and fitness evaluators.

The architectural design of a DNN model can be decomposed into the design of (i) the skeleton of the network (i.e., the depth and width of the network) and (ii) the layer (i.e., the types and arrangements of the operators such as convolution, pooling, etc.). Early work made attempts to design these two aspects simultaneously, resulting in DNN architectures with sub-optimal performance [7], [14], [37]. Accordingly, recent efforts resort to only one aspect, and two types of search spaces have been proposed. Specifically, *micro* search spaces focus on designing a modular computational block (also known as a *cell*), which is repeatedly stacked to form a complete DNN architecture following a pre-specified template [15]–[18]; *macro* search spaces focus on designing the network skeleton while leaving layers to well-established designs [38]–[40]. A pictorial illustration is provided in Fig. 2.

On one hand, micro search spaces confine the search to the inner configuration of a layer, leading to a considerable reduction in search space volume at the cost of structural diversity among layers, which has been shown to be crucial for hardware efficiency [41]. On the other hand, despite compelling performance on hardware, macro search spaces are oftentimes criticized for producing architecturally similar DNN models [42].

C. Existing NAS Benchmarks

As one of the earliest tabular benchmarks[1], NAS-Bench-101 [21] adopts a micro search space where approximately 423K unique cell structures are exhaustively evaluated three times on the CIFAR-10 dataset [25] for image classification.

[1]A tabular benchmark means that all attainable solutions are exhaustively evaluated and stored as a tabular dataset a priori.

All results are stored in a *TFRecord* file, which is a binary file format designed specifically for TensorFlow—a dedicated software for machine learning with DNNs [43]. A subsequent tabular benchmark of NAS-Bench-201 [22], with a similar micro search space of 16K cell structures, extends the dataset to include the CIFAR-100 [25] and downsampled ImageNet 16×16 [44] datasets. Moving beyond tabular benchmarks, NAS-Bench-301 introduces the first surrogate-model-based benchmark on the CIFAR-10 dataset. With sophisticated regression models (e.g., graph isomorphism network [45] and tree-based gradient boosting methods [46], [47]) on top of a large number of training samples (∼60K), it extends the volume of a micro search space to include more than 10^{21} cell structures. Both NAS-Bench-201 and -301 are implemented in PyTorch, another sophisticated deep learning software [48], with additional dependencies for running sophisticated surrogate models.

Concurrently, a steady stream of follow-up work have been reported, extending NAS benchmarks to include: (i) a macro search space for finding the optimal channels of each layer in a DNN (i.e., NATS-Bench [23]), (ii) hardware-related performance (HW-NAS-Bench [49]), (iii) or other application domains such as natural language processing [50], speech recognition [51], transfer learning [52], among others.

III. Problem Formulation

Given a target dataset $\mathcal{D} = \{\mathcal{D}_{trn}, \mathcal{D}_{vld}, \mathcal{D}_{tst}\}$ and target hardware device set $\mathcal{H} = \{h_1, \ldots, h_{|\mathcal{H}|}\}$, without loss of generality, a NAS task can be formulated as a multi-objective optimization problem:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{F}(\boldsymbol{x}) = \Big(f^e(\boldsymbol{x}; \boldsymbol{w}^*(\boldsymbol{x})), \boldsymbol{f}^c(\boldsymbol{x}), \boldsymbol{f}^{\mathcal{H}}(\boldsymbol{x})\Big) \\ \text{s.t.} \quad & \boldsymbol{w}^*(\boldsymbol{x}) \in \operatorname{argmin} \ \mathcal{L}_{trn}(\boldsymbol{x}; \boldsymbol{w}), \quad \boldsymbol{x} \in \Omega \end{aligned} \tag{2}$$

with

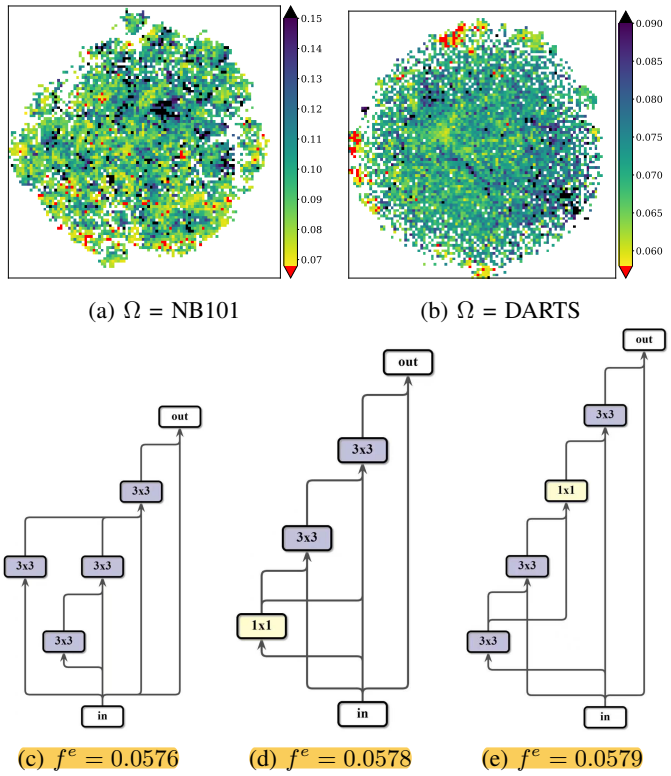
$$\boldsymbol{f}^c(\boldsymbol{x}) : f_1^c(\boldsymbol{x}), \cdots, f_{M^c}^c(\boldsymbol{x}), \tag{3}$$

and

$$\boldsymbol{f}^{\mathcal{H}}(\boldsymbol{x}) : \begin{cases} f_1^{h_1}(\boldsymbol{x},), \cdots, f_{M_1^h}^{h_1}(\boldsymbol{x}), \\ \quad \vdots \\ f_1^{h_{|\mathcal{H}|}}(\boldsymbol{x},), \cdots, f_{M_{|\mathcal{H}|}^h}^{h_{|\mathcal{H}|}}(\boldsymbol{x}) \end{cases}, \tag{4}$$

where Ω is the search space, $\boldsymbol{x}$ and $\boldsymbol{w}(\boldsymbol{x})$ denote an encoded network architecture (i.e. decision vector) and the corresponding weights of the decoded network respectively, and $\boldsymbol{w}^*(\boldsymbol{x})$ denotes the optima weights achieving the minimal loss on the training set $\mathcal{D}_{trn}$; f^e denotes the objective function indicating the *prediction error* of the model, $\boldsymbol{f}^c$ and $\boldsymbol{f}^{\mathcal{H}}$ denote the objective functions indicating the performance related to *model complexity* and *hardware devices* respectively.

In practice, the objective functions can be formulated in various ways. For example, to formulate f^e, the most straightforward approach is to associate it with the validation loss on $\mathcal{D}_{vld}$; to formulate $\boldsymbol{f}^c$, we usually refer to the number of parameters (i.e. weights) and floating point of operations (FLOPs) respectively; to formulate $\boldsymbol{f}^c$, the most commonly considered metrics are hardware latency and hardware energy consumption. Besides, since it can be computationally prohibitive to fully train a model on $\mathcal{D}_{trn}$ to obtain the exact validation loss on $\mathcal{D}_{vld}$ for f^e, surrogate-modeling approaches are also adopted.



(a) Ω = NB101 (b) Ω = DARTS

(c) $f^e = 0.0576$ (d) $f^e = 0.0578$ (e) $f^e = 0.0579$

Fig. 3: Fitness landscapes of f^e (i.e. prediction error) on (a) NB101 and (b) DARTS. We project the original high-dimensional decision space to 2D latent space via t-SNE [53]. We random sample 10K solutions from each search space and average f^e within each small area. The top-performing solutions are highlighted in red. (c)-(e) are three solutions with similar f^e but different decision vectors (i.e., architectures) sampled from the NB101 search space.

Based on the formulation above, the following subsections will elaborate on the complex characteristics of NAS tasks from the optimization point of view.

A. Multi-modal Fitness Landscapes

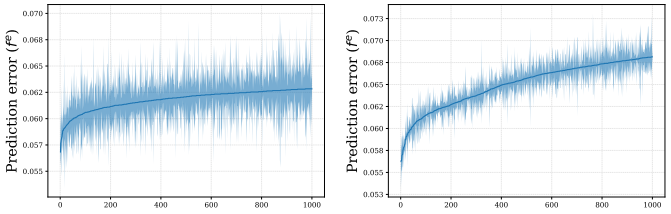
The fitness landscape of an optimization problem depicts how fitness values change over the decision space. Specifically, a *multi-modal* landscape refers to the case involving multiple optimum solutions (a.k.a multi-modality) having very close (or even the same) fitness values [54]. When searching over a multi-modal fitness landscape, the target is often to find the multiple optimum solutions once, such that the decision-maker is able to choose among those solutions according to personal preferences. Particularly, there are emerging research interests in studying multi-modal MOPs in the EMO community [55].

In the context of NAS, it is very likely that different network architectures can lead to very close (or even the same) prediction accuracy or hardware-related metrics such as latency. As evidenced in Fig. 3, for example, the fitness

landscapes of f^e on search spaces NB101 and NB201 contain a number of optima having very close fitness values (i.e., prediction errors); correspondingly, the solutions having very close fitness values could represent very different architectures.

B. Noisy Objectives

In practical optimization problems, a solution is often evaluated through stochastic simulations, physical experiments, or even interactions with users. As a result, the outputs for repeated evaluations at the same decision vector are not deterministic. From an optimization point of view, we characterize such an evaluation process as a *noisy objective* [56], [57]. Evolutionary computation is believed to be well suited for tackling noisy objectives given its population-based nature and randomized search heuristics. Accordingly, developing effective algorithms for solving noisy optimization problems has always been an active research topic in the EMO community [58], [59].



(a) Ω = NB101 (b) Ω = NB201

Fig. 4: The mean f^e (prediction error) of the top 1,000 solutions (according to f^e) on (a) NB101 and (b) NB201 search spaces from three repetitions. The standard deviations (i.e., *noise*) of f^e are visualized with the shaded error bars.

In the context of NAS, the evaluation of f^e (prediction error) of an architecture is almost always noisy due to various stochastic components involved in training the corresponding weights of the architecture, such as randomized data augmentation and loading order. Fig. 4 provides examples visualizing the noises in evaluating f^e on NB101 and NB201 search spaces respectively. Furthermore, hardware-related objectives also critically depend on the operating conditions of the hardware, such as current loading, ambient temperature, etc. Hence, the evaluation of $\boldsymbol{f}^{\mathcal{H}}$ is also subject to high variance.

C. Many Objectives

The number of objectives in a MOP is an important character to be considered in EMO. Particularly, if the number of objectives increases to more than three, an MOP is known to be a *many-objective optimization problem (MaOP)*, posing great challenges to the EMO algorithms [60]. On the one hand, it is challenging to obtain the sparsely distributed candidate solutions when considering both convergence and diversity in the high-dimensional objective space; on the other hand, it is challenging to perform decision-making when trading off among many objectives.

As formulated above, a NAS task can involve up to a number of $1 + M^c + \sum_{i=1}^{|\mathcal{H}|} M_i^h$ objective functions to be optimized simultaneously, where M_c denotes the number of objective functions related to model complexity, and M_i^h denotes the number of objective functions related to the i-th hardware device in $\mathcal{H}$. As exemplified in Fig. 5, a NAS task on the search space of NB201 consists of six objectives with $M^c = 2$ and $M_1^h = 3$.

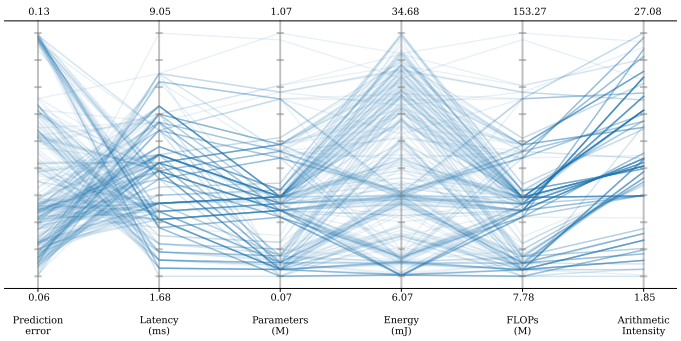


Fig. 5: The parallel coordinates of non-dominated solutions of architectures sampled from the NB021 search space with hardware device of Eyeriss (i.e., Ω = MNV3, $\mathcal{H} = \{h_1 = \text{Eyeriss}\}$). The six optimization objectives: f^e: prediction error, f_1^c: number of parameters (M), f_2^c: number of FLOPs (M), $f_1^{\mathcal{H}}$: latency (ms), $f_3^{\mathcal{H}}$: energy (mJ), $f_3^{\mathcal{H}}$: Arithmetic Intensity (ops/byte).

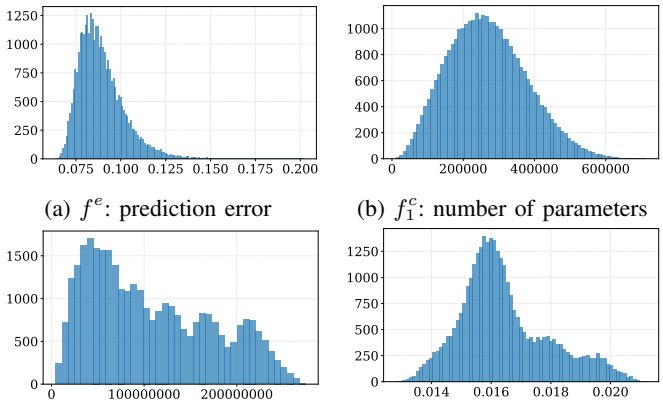
D. Badly Scaled Objectives

Ideally, the objective functions of a MOP should be scaled to the same (or similar) values such that EMO algorithms could work effectively. This is particularly important to the decomposition-based EMO algorithms whose performances largely rely on the predefined weight/reference vectors. Without any preference or *a priori* knowledge, the weight/reference vectors are uniformly sampled from a unit hyper-plan/sphere, assuming that the objective functions of the MOP to be solved are also well normalized. In practice, however, the objective functions can be of very different scales and there is no way to formulate them in a normalized manner. An MOP of such a character is often known to be *badly scaled* [61].

Considering the three types of objective functions in a general formulation of a NAS task, the corresponding MOPs are intrinsically badly scaled. As exemplified in Fig. 6, the prediction error (f^e) falls into the range of $[0, 1]$; by contrast, since the complexity related objectives ($\boldsymbol{f}^c$) and hardware related objectives ($\boldsymbol{f}^{\mathcal{H}}$) are formulated to indicate different physical quantities, their objective values are of very different scales. Moreover, it is difficult to normalize these objective values into the same scale due to the unknown properties of various search spaces and hardware devices.

E. Degenerate Pareto fronts

In multiobjective optimization, an important assumption is that the objectives to be optimized are often in conflict, such that there does not exist a single solution achieving optima on all the objectives. Alternatively, an EMO algorithm aims to find a set of solutions as an approximation to the Pareto

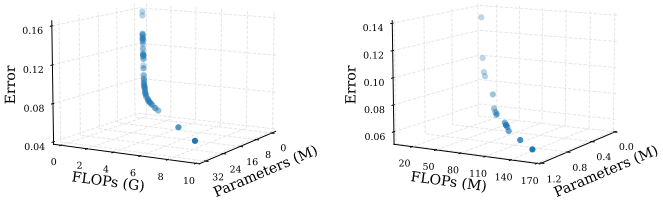


(a) f^e: prediction error (b) f_1^c: number of parameters

(c) f_2^c: number of FLOPs (d) $f_1^{\mathcal{H}}$: latency (seconds)

Fig. 6: Histogram of the objective values of architectures sampled from the NATS search space with hardware device of GPU (i.e., Ω = NATS, $\mathcal{H} = \{h_1 = \text{GPU}\}$). The x-axis of each subfigure shows the *range* of the objective values, while the y-axis shows the number of corresponding architectures.

front (PF), where the solutions on the PF trade-off between the conflicting objective. Ideally, the Pareto front (PF) of an m-objective MOP is an $m-1$-dimensional manifold iff all the objectives are conflicting with each other. In practice, however, this property may not always hold since some objectives of a MOP may be positively correlated to each other, thus leading to a *degenerate PF* [62], [63].



(a) $\Omega = $ NB101 (b) $\Omega = $ NB201

Fig. 7: The Pareto fronts of two three-objective NAS tasks on the (a) NB101 and (b) NB201 search spaces. The three optimization objectives are: f^e: prediction error, f_1^c: number of parameters, f_2^c: number of FLOPs.

As formulated above, there are generally three types of objectives in a NAS problem: the *error*-related objective $\boldsymbol{f}^e$, the *complexity*-related objectives $\boldsymbol{f}^c$,and the *hardware*-related objectives $\boldsymbol{f}^{\mathcal{H}}$. Normally, f^e is conflicting with $\boldsymbol{f}^c$ and $\boldsymbol{f}^c$ respectively, i.e., a model with lower prediction error usually has higher complexity and lower hardware performance. By contrast, however, the objectives in $\boldsymbol{f}^c$ or $\boldsymbol{f}^{\mathcal{H}}$ may not be fully conflicting among themselves. As exemplified in 7a, f^e is conflicting with f_1^c and f_2^c respectively, while f_1^c and f_2^c are positively correlated, thus leading to a degenerate PF (i.e. the one-dimensional curve) in the three objective space.

IV. Benchmark Design

In this section, we introduce the overall pipeline of `EvoXBench`, followed by detailed design principles of each main component.

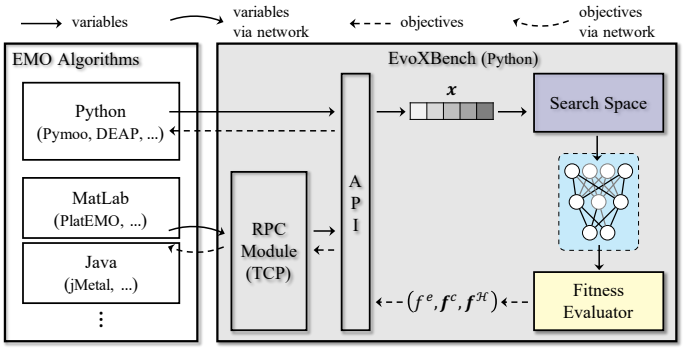


Fig. 8: Overall pipeline of `EvoXBench`. See text for details.

A. Overview

In order to facilitate seamless communication, an API layer is designed to handle (i) the reception of the candidate decision vectors generated by an EMO algorithm and (ii) the feedback of the fitness values evaluated by our `EvoXBench`. As shown in Fig. 8, given a candidate decision vector received by the API layer, `EvoXBench` first passes it through the search space module to create its corresponding architecture; then, the architecture is processed by the fitness evaluator module; finally, the optimization objectives are routed back to the API layer for output. For EMO algorithms implemented in non-Python programming languages (e.g., MATLAB or Java), the communication is routed through an extra remote procedure call (RPC) module built on top of the widely-used transmission control protocol (TCP).

B. Design of Search Spaces

As formulated in (2), a search space (a.k.a. decision space) Ω defines all attainable solutions to an optimization algorithm. In general, every search space follows the schema (outlined in Algorithm 1 from supplementary materials). Specifically, a string that defines a DNN architecture is referred to as a *phenotype*, from which an actual DNN model can be created; an integer-valued vector is referred to as a *genotype* on which genetic operators, such as crossovers and mutations, are carried out; and the interfaces between genotypes and phenotypes are referred to as *encode* and *decode*, respectively.

In `EvoXBench`, we currently consider seven search spaces for image classification on CIFAR-10 and ImageNet datasets, covering both convolutional DNNs and Transformers from micro and macro search spaces. A summary of these search spaces is provided in Table II, and visualizations of architectures from these search spaces are provided in Fig. 9.

It is worth noting that, apart from the existing seven search spaces, any other user-specific search space can also be easily incorporated into `EvoXBench` by following a routine pipeline. Due to the page limit, readers are referred to Section IV in the supplementary materials for more details.

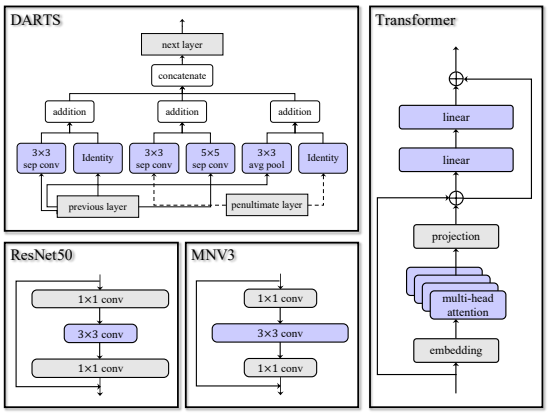


Fig. 9: Visualization of architectures from search spaces supported in EvoXBench.

TABLE II: An overview of the search spaces supported in EvoXBench. The fitness landscapes of the search spaces from the top section are completely known via exhaustive evaluations, while the fitness landscapes of the search spaces from the bottom section are approximated via surrogate models trained on $60K$ uniformly sampled architectures.

Ω	Type	Dataset	D	$\lvert\Omega\rvert$	Objectives
NB101 [21]	micro	C-10	26	423K	f^e, $\boldsymbol{f}^c$
NB201 [22], [49]	micro	C-10	6	15.6K	f^e, $\boldsymbol{f}^c$, $\boldsymbol{f}^{h_1}$
NATS [23]	macro	C-10	5	32.8K	f^e, $\boldsymbol{f}^c$, f^{h_1}
DARTS [24]	micro	C-10	32	$\sim 10^{21}$	f^e, $\boldsymbol{f}^c$
ResNet50 [64]	macro	IN-1K	25	$\sim 10^{14}$	f^e, $\boldsymbol{f}^c$
Transformer [65]	macro	IN-1K	34	$\sim 10^{14}$	f^e, $\boldsymbol{f}^c$
MNV3 [64]	macro	IN-1K	21	$\sim 10^{20}$	f^e, $\boldsymbol{f}^c$, f^{h_1}

C. Design of Fitness Evaluator

As formulated in (2), we consider three categories of objectives, i.e., the prediction error objective (f^e), the model complexity related objectives ($\boldsymbol{f}^c$) and hardware efficiency ($\boldsymbol{f}^{\mathcal{H}}$). Dedicated methods are devoted to handling these objectives efficiently and reliably.

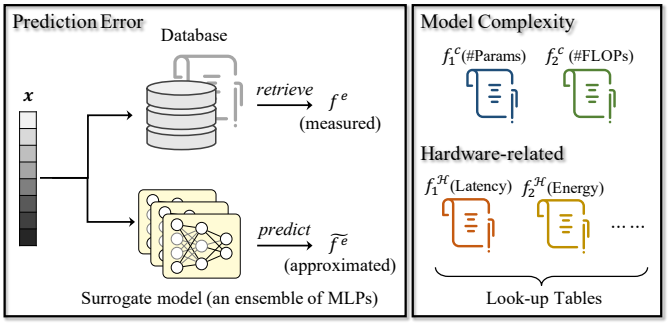


Fig. 10: Overview of fitness evaluators in EvoXBench.

1) Evaluation of f^e: As illustrated in Section III-B, f^e is a noisy optimization objective. To simulate the noise in evaluating f^e, the following two strategies are employed:

– For NB101, NB201, and NATS search spaces, we leverage the exhaustive evaluation results provided by the original papers [21]–[23], [49], where all solutions (architectures) are thoroughly trained from scratch with three repetitions. On top of these results, we construct a unified database solely based on Python. In other words, the f^e values of solutions from these search spaces can be queried via a canonical interface without configurations of sophisticated software such as TensorFlow or PyTorch. Thereafter, each evaluation value of f^e is randomly selected from the three repetitions of the corresponding evaluation results stored in the database.

– Considering the total volume of the DARTS, ResNet50, Transformer, and MNV3 search spaces, we resort to surrogate models for evaluating f^e. We choose a multi-layer perceptron (MLP) as our surrogate model in this work. Apart from its well-established track record for predicting the f^e values of DNNs accurately [64], [66], [67], implementing an MLP (i.e., a loop of matrix multiplications and additions) is straightforward and canonical in most programming languages without additional dependencies. Furthermore, we construct a pool of MLPs from k-fold cross-validation of the training data, where k is set to ten in this work. Thereafter, each evaluation value of f^e is predicted by a single MLP randomly selected from the pool.

For generating samples (i.e., variable-objective pairs) to train a MLP, we utilize the *supernet* idea that has emerged as a standard technique in state-of-the-art (SotA) NAS methods [42]. Specially, first, we follow the progressive shrinking algorithm [64] to train a supernet from which the optimal weights of a candidate architecture solution (i.e., $\boldsymbol{w}^*(\boldsymbol{x})$ in Eq 2) can be directly inherited without the costly iterations of SGD[2]; then, we use the weights provided by the trained supernet to evaluate the f^e values of the samples. As opposed to generating samples by independently training every one of them from scratch, this pipeline offers two appealing properties: (i) it is more scalable to a large number of samples which is crucial for learning an accurate approximation[3]; (ii) the post-search re-training of the obtained architectures can be avoided as the corresponding weights are readily available from the supernet.

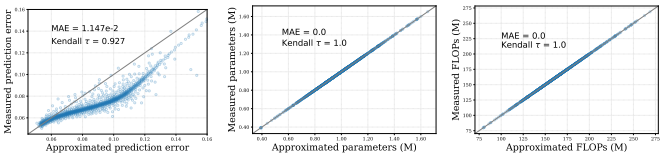
A pictorial illustration of the f^e evaluators in EvoXBench is provided in the top part of Fig. 10. And the performance of the adopted surrogate model, an ensemble of MLPs, is provided in Fig. 11.

2) Evaluation of $\boldsymbol{f}^c$ and $\boldsymbol{f}^{\mathcal{H}}$: Similar to the evaluation of f^e, we also leverage the exhaustive evaluation history provided by the original papers [21]–[23], [49] for NB101, NB201 and NATS search spaces. These $\boldsymbol{f}^c$ and $\boldsymbol{f}^{\mathcal{H}}$ values are, again, stored in a unified database along with f^e values.

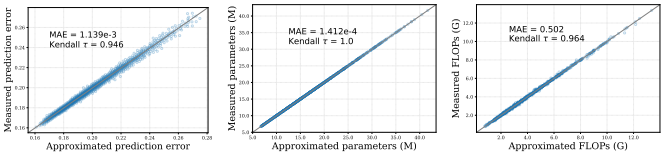
For the DARTS, ResNet50, Transformer, and MNV3 search spaces, we opt for the *loop-up table* route—a standard approach among existing NAS methods where an exhaustive exploration of the search space is not possible [39], [49]. A pictorial overview is provided in the bottom part of Fig. 10.

[2]SGD is a standard method for solving the inner optimization problem in Eq 2, i.e., $\min\ \mathcal{L}_{trn}(\boldsymbol{x}; \boldsymbol{w})$.

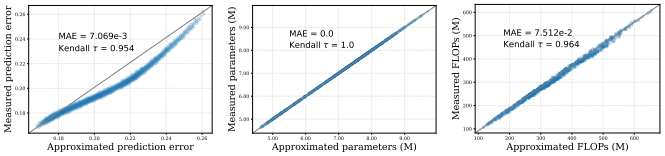
[3]Assuming that the training cost of a large number of sample architectures, $60K$ in our case, is much greater than the training cost of a single supernet.



(a) DARTS search space



(b) ResNet50 search space



(c) MNV3 search space

Fig. 11: Performance of our fitness evaluators. For each search space, we visualize the correlation between real measurements and approximations by our surrogate model for prediction error (f^e), look-up tables for the number of parameters (f_1^c) and FLOPs (f_2^c), from left to right. The mean absolute error (MAE) and Kendall rank correlation values are annotated in every sub-figure.

In general, on one hand, the efficiency of a cell structure can be decomposed into the accumulative efficiency of the participated operations for a micro search space (i.e., DARTS); while on the other hand, the efficiency of a network can be decomposed into the accumulative efficiency of each layer for macro search spaces (i.e., ResNet50, Transformer, and MNV3).

Under such assumptions, there exist limited operations and layer variations to enumerate, respectively for micro and macro search spaces. Hence, we can exhaustively evaluate the $\boldsymbol{f}^c$ and $\boldsymbol{f}^{\mathcal{H}}$ values of all operations and layer variations. Afterward, the efficiency of any given operation or layer variation can be queried from a look-up table with a designated key. Then the $\boldsymbol{f}^c$ and $\boldsymbol{f}^{\mathcal{H}}$ values of an architecture solution can be obtained by summing over all operations or layer variations.

D. Design of RPC Module

As depicted in Fig. 8, EMO algorithms written in Python can call `EvoXBench` directly, while queries from EMO algorithms in other programming languages are processed through the remote procedure call (RPC) module. For compatibility reasons, the communication functionalities of RPC are built upon the transmission control protocol (TCP). Technically, it allows programs written in almost any programming language on the operating system to call `EvoXBench` via localhost. Furthermore, by setting up `EvoXBench` once on a central server, it allows additional users to use `EvoXBench` remotely as a web service without any local installation/configuration.

More specifically, after establishing a successful TCP connection, EMO algorithms (as the users) and `EvoXBench` communicate by exchanging JSON strings. Since this protocol itself is stateless, we assign an integer string to every object created in `EvoXBench` as its unique identifier. And this identifier needs to be included as a part of the JSON strings to facilitate the communication between users and `EvoXBench`.

As depicted in Table III, we empirically demonstrate that real-time feedback of fitness values is archived by `EvoXBench` via the RPC module. Specifically, the evaluation of an architecture in `EvoXBench` is facilitated by either querying a database or a surrogate model. On one hand, querying a database is intrinsically fast; on the other hand, the surrogate model used in `EvoXBench` is a simple yet effective three-layer perceptron. Additionally, we batch all evaluations in a matrix to further boost the speed. Readers may refer to supplementary materials for more technical details.

TABLE III: Total time cost (in **seconds**) for calling `EvoXBench` to evaluate **1,000** solutions by EMO algorithms implemented in three different programming languages on a local machine. Results are averaged over 31 runs on a single **CPU** core.

Search space	Query method	Python	MATLAB	Java
NB101	Database	0.1139 ± 0.013	0.1716 ± 0.031	0.1970 ± 0.036
MNV3	Surrogate	0.0380 ± 0.002	0.0528 ± 0.018	0.0574 ± 0.020

V. Benchmark Test Suite Generation

On the basis of `EvoXBench`, specific test suites (i.e., collections of test instances/problems) can be generated by combining search spaces and their supported objectives. For the purpose of demonstration, we generate two tailored multi-objective NAS test suites for image classification on datasets CIFAR-10 and ImageNet respectively. Following the formulations in Section III, the test instances are defined by specifying the search space, the hardware, and the metrics for measuring model complexity and hardware efficiency. In the remaining of this section, we present these two test suites in detail.

A. C-10/MOP Test Suite

As summarized in Table IV, there are nine instances in C-10/MOP tailored for image classification on CIFAR-10, considering various search spaces and hardware devices. The test instances are arranged in the order of ascending number of objectives, i.e., from two to eight objectives. For most test instances, we consider only one targeted hardware (i.e., GPUs or Eyeriss [68]) except C-10/MOP7 where both hardware are considered simultaneously.

The properties of the test instances are summarized in Table V in correspondence with the characters illustrated in Section III. In general, the fitness landscapes of prediction error (f^e) are noisy and multi-modal, and the Pareto fronts are degenerated for most test instances. Since the fitness landscape of C-10/MOP1 to C-10/MOP7 are completely known via exhaustive samples, the ranges of all objectives are known *a priori* and thus can be properly scaled By contrast, the objectives of C-10/MOP8 and C-10/MOP9 remain un-normalized (i.e., badly scaled) as the ranges cannot be estimated via exhaustive samples.

TABLE IV: Definition of the proposed C-10/MOP test suite.

Problem	Ω	D	M	Objectives
C-10/MOP1	NB101	26	2	f^e, f_1^c
C-10/MOP2	NB101	26	3	f^e, f_1^c, f_2^c
C-10/MOP3	NATS	5	3	f^e, f_1^c, f_2^c
C-10/MOP4	NATS	5	4	f^e, f_1^c, f_2^c, $f_1^{h_1}$
C-10/MOP5	NB201	6	5	f^e, f_1^c, f_2^c, $f_1^{h_1}$, $f_2^{h_1}$
C-10/MOP6	NB201	6	6	f^e, f_1^c, f_2^c, $f_1^{h_2}$, $f_2^{h_2}$, $f_3^{h_2}$
C-10/MOP7	NB201	6	8	f^e, f_1^c, f_2^c, $f_1^{h_1}$, $f_2^{h_1}$, $f_1^{h_2}$, $f_2^{h_2}$, $f_3^{h_2}$
C-10/MOP8	DARTS	32	2	$\dagger f^e$, f_1^c
C-10/MOP9	DARTS	32	3	$\dagger f^e$, f_1^c, f_2^c

† indicates that f^e (validation error) is based on surrogate modeling.
‡ hardware h_1 = GPU and h_2 = Eyeriss.

TABLE V: Property of test instances in C-10/MOP.

Problem	Multi-modal	Many objectives	Noisy objectives	Badly-scaled	Degenerated PF
C-10/MOP1	✓		✓		
C-10/MOP2	✓		✓		✓
C-10/MOP3					✓
C-10/MOP4		✓			✓
C-10/MOP5	✓	✓	✓		✓
C-10/MOP6	✓	✓	✓		✓
C-10/MOP7	✓	✓	✓		✓
C-10/MOP8	✓		✓	✓	
C-10/MOP9	✓		✓	✓	

B. IN-1K/MOP Test Suite

The IN-1K/MOP test suite also comprises nine test instances. Since the search spaces of ResNet50 and Transformer are resource-intensive and thus unsuitable for efficient hardware deployment, we do not consider hardware-related objectives as architectures from them. As a result, most test instances are multiobjective problems except IN-1K/MOP9 where an additional objective of the latency on a mobile phone is considered. A summary of IN-1K/MOP test suite is provided in Table VI.

TABLE VI: Definition of the proposed IN-1K/MOP test suite.

Problem	Ω	D	M	Objectives
IN-1K/MOP1	ResNet50	25	2	f^e, f_1^c
IN-1K/MOP2	ResNet50	25	2	f^e, f_2^c
IN-1K/MOP3	ResNet50	25	3	f^e, f_1^c, f_2^c
IN-1K/MOP4	Transformer	34	2	f^e, f_1^c
IN-1K/MOP5	Transformer	34	2	f^e, f_2^c
IN-1K/MOP6	Transformer	34	3	f^e, f_1^c, f_2^c
IN-1K/MOP7	MNV3	21	2	f^e, f_1^c
IN-1K/MOP8	MNV3	21	3	f^e, f_1^c, f_2^c
IN-1K/MOP9	MNV3	21	4	f^e, f_1^c, f_2^c, $f_1^{h_1}$

† All f^e (validation errors) are based on surrogate modeling.
‡ hardware h_1 = Samsung Note10.

The properties of the test instances of IN-1K/MOP are summarized in Table VII. Given the volume of the involved search spaces, it is impossible to evaluate every attainable solution. Accordingly, for each test instance, the fitness landscape of is approximated via a surrogate model, and the objectives are badly-scaled (i.e., without normalization). Note that the objectives can be normalized to the bounds derived from the sampled architectures (for building the surrogate models). However, due to the fact that the sampled architectures may not exactly cover the full range of the search spaces, the "normalized" objectives (according to the bounds derived from the sampled architectures) may not be strictly between zero and one. Accordingly, we opt for leaving the objectives as un-normalized. The character of multi-modality for all test instances is verified on the basis of extensive optimization using a large number of fitness evaluations.

TABLE VII: Property of test instances in IN-1K/MOP.

Problem	Multi-modal	Many objectives	Noisy objectives	Badly-scaled
IN-1K/MOP1	✓		✓	✓
IN-1K/MOP2	✓		✓	✓
IN-1K/MOP3	✓		✓	✓
IN-1K/MOP4			✓	✓
IN-1K/MOP5			✓	✓
IN-1K/MOP6			✓	✓
IN-1K/MOP7	✓		✓	✓
IN-1K/MOP8	✓		✓	✓
IN-1K/MOP9	✓	✓	✓	✓

VI. Experimental Study

In this section, we provide an experimental study using the two test suites proposed in the previous section. First, we explain the experimental setup, including the selected EMO algorithms and the performance metric; then, we present experimental results followed by a discussion of the observations. Constrained by space, readers are referred to supplementary materials for implementation details.

A. Experimental Settings

A plethora of algorithms have been proposed in the EMO literature for solving MOPs (involving two/three objectives) or MaOPs (involving more than three objectives). Accordingly, we select six representative algorithms from each category as introduced in Section II-A, including: NSGA-II [11] (dominance-based), IBEA [30] (indicator-based), MOEA/D [12] (decomposition-based), NSGA-III [69] (dominance-based), HypE [70] (indicator-based), and RVEA[4] [71] (decomposition-based), where the first three were classic ones for solving MOPs and the rest were tailored for solving MaOPs.

TABLE VIII: Population size settings.

M	(H_1, H_2)	N
2	(99, 0)	100
3	(13, 0)	105
4	(7, 0)	120
5	(5, 0)	126
6	(4, 1)	132
8	(3, 2)	156

All experiments are conducted on PlatEMO [20] – an EMO algorithm library implemented in MATLAB. The population size is set in correspondence with the number objectives, as shown in Table VIII. We perform 31 independent runs for each algorithm on each test instance using 10,000 fitness evaluations, and the statistical results are compared using Wilcoxon rank sum test.

[4]We use the version with the reference vector regeneration strategy, which is also known as the RVEA*.

TABLE IX: Statistical results (median and standard deviation) of the HV values on C-10/MOP test suite. The best results of each instance are in bold.

Problem	NSGA-II	IBEA	MOEA/D	NSGA-III	HypE	RVEA
C-10/MOP1	**0.9367 (0.0042)**$^{\approx}$	0.8627 (0.0186)$^{-}$	0.9069 (0.0151)$^{-}$	**0.9379 (0.0047)**$^{\approx}$	0.8488 (0.0678)$^{-}$	0.9297 (0.0083)$^{-}$
C-10/MOP2	**0.9176 (0.0025)**$^{+}$	0.8341 (0.0228)$^{-}$	0.8727 (0.0064)$^{-}$	0.7108 (0.2048)$^{-}$	0.7879 (0.0848)$^{-}$	0.9091 (0.0071)$^{-}$
C-10/MOP3	0.8235 (0.0015)$^{-}$	**0.8323 (0.0009)**$^{+}$	0.8000 (0.0047)$^{-}$	0.8083 (0.0044)$^{-}$	0.8183 (0.0071)$^{-}$	0.8262 (0.0072)$^{-}$
C-10/MOP4	0.7656 (0.0085)$^{-}$	**0.7941 (0.0061)**$^{+}$	0.7277 (0.0125)$^{-}$	0.7633 (0.0091)$^{-}$	0.7692 (0.0080)$^{-}$	0.7544 (0.0189)$^{-}$
C-10/MOP5	0.7127 (0.0000)$^{-}$	0.7094 (0.0007)$^{-}$	0.6721 (0.0045)$^{-}$	0.7040 (0.0179)$^{-}$	**0.7569 (0.0017)**$^{+}$	0.7521 (0.0077)$^{-}$
C-10/MOP6	0.7404 (0.0001)$^{-}$	0.7302 (0.0005)$^{-}$	0.6807 (0.0560)$^{-}$	0.6387 (0.0149)$^{-}$	**0.7743 (0.0006)**$^{+}$	0.7549 (0.0145)$^{-}$
C-10/MOP7	0.5696 (0.0118)$^{-}$	0.5892 (0.0012)$^{-}$	0.5108 (0.0509)$^{-}$	0.5417 (0.0202)$^{-}$	**0.6322 (0.0068)**$^{+}$	0.6122 (0.0146)$^{-}$
C-10/MOP8	**0.9769 (0.0043)**$^{+}$	0.9751 (0.0080)$^{-}$	0.8870 (0.0250)$^{-}$	0.9741 (0.0064)$^{-}$	0.9484 (0.0115)$^{-}$	0.9185 ((0.0168)$^{-}$
C-10/MOP9	**0.9630 (0.0097)**$^{+}$	0.9609 (0.0136)$^{-}$	0.7633 (0.0404)$^{-}$	0.9579 (0.0066)$^{-}$	0.9253 (0.0085)$^{-}$	0.8952 (0.0178)$^{-}$

$^{+}$ indicates a method achieving significantly better performance.
$^{\approx}$ indicates a method achieving similar performance as the best-performing method.
$^{-}$ indicates a method achieving significantly worse performance.

B. Performance Indicator

In this paper, we adopt hypervolume (HV) to quantitatively compare the performance among the considered algorithms. Let us denote $\boldsymbol{y}^{\text{ref}} = (y_1, \ldots, y_m)$ as a reference point dominated by all Pareto-optimal solutions in the objective space, and $\boldsymbol{Y}$ as the Pareto-front approximated by an algorithm. The HV value of $\boldsymbol{Y}$ (with respect to $\boldsymbol{y}^{\text{ref}}$) is the volume of the region dominating $\boldsymbol{y}^{\text{ref}}$ and dominated by $\boldsymbol{Y}$.

Specifically, we follow two rules to set the reference point for calculating HV: (i) for problems derived from search spaces that are exhaustively evaluated (i.e., $\boldsymbol{\Omega} = \{\text{NB101, NB201, NATS}\}$), we set $\boldsymbol{y}^{\text{ref}}$ to the nadir point since the actual Pareto front is available; (ii) for problems derived on the basis of surrogate models (i.e., $\boldsymbol{\Omega} = \{\text{DARTS, MNV3, ResNet50, Transformer}\}$), we set $\boldsymbol{y}^{\text{ref}}$ to the worst point among the samples collected for training the surrogate models. Readers are referred to Table A.I in the supplementary materials for the specific reference point for each test instance.

C. Results

In the following, we present results achieved by each of the considered algorithm on the two test suites.

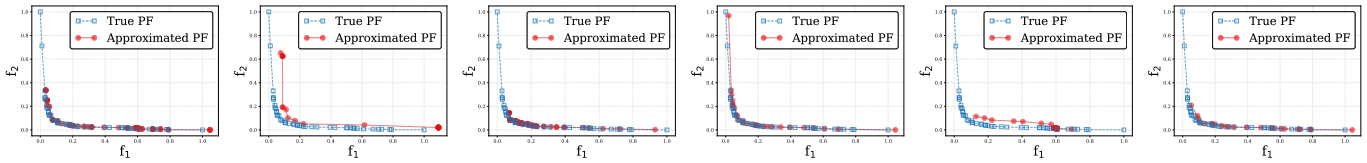
1) Results on C-10/MOP: Table IX summarizes the statistical values of the HV metric achieved by the six algorithms. In general, we can observe that none of the six algorithms can effectively solve all instances in the C-10/MOP test suite. More specifically, on MOPs with two or three objectives (i.e., C-10/MOP1, C-10/MOP2, C-10/MOP8, and C-10/MOP9), we can observe that NSGA-II consistently outperforms other algorithms, except C-10/MOP3 where IBEA performs slightly better, while on MaOPs involving more than three objectives (i.e., C-10/MOP4 - C-10/MOP7), we can observe that HypE yields generally the best performance For further observations, we visualize the final nondominated solutions obtained by each algorithm in the median run on two typical test instances in Fig. 12a and Fig. 12b for C-10/MOP1 and C-10/MOP6 respectively. In the following, we provide some discussions based on the observations made from Fig. 12.

C-10/MOP1 is a problem with a simple bi-objective convex PF; nevertheless, given its multi-modal and noisy fitness landscape, the problem is still non-trivial to solve. First, we can observe that all algorithms fail to converge to the low-f_1 regime (top-left corner), indicating the challenges for obtaining nondominated solutions with low f_1 (i.e., f^e; prediction error). Second, we can observe that indicator-based algorithms (i.e., IBEA and HypE) perform significantly worse than other peer methods. Third, both NSGA-II and NSGA-III achieve the best performance, where NSGA-II has slightly better coverage of the low-f_2 regime while NSGA-III has a slightly better converge of the low-f_1 regime.

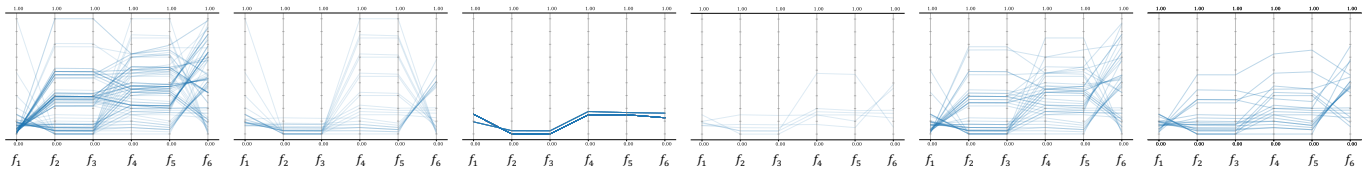
C-10/MOP6 is relatively more complex with six (but partially correlated) objectives. The objective function of f_1 (i.e., f^e; prediction error) is also multi-modal and noisy, but the number of variables is small, i.e., six. First, we can observe that objective 2 (f_1^c; No. of parameters) and objective 3 (f_2^c: No. of FLOPs), objective 4 ($f_1^{h_2}$: latency on Eyeriss), and objective 5 ($f_2^{h_2}$: energy consumption on Eyeriss) are correlated, thus leading to a degenerated PF. Second, we can observe that the decomposition-based algorithms (i.e., MOEA/D and RVEA) are significantly worse than those with purely Pareto-dominance-based (i.e., NSGA-II) or indicator-based (i.e., IBEA and HypE) ones.

2) Results on IN-1K/MOP: This test suite comprises primarily bi-/three-objective MOPs except for the last one – IN-1K/MOP9 with four objectives. The statistical values of the HV metric achieved by the six algorithms are summarized in Table X. In general, similar to the results of the previous test suite, we can observe that none of the six algorithms can effectively solve all problems in IN-1K/MOP. In particular, on bi-objective problems (i.e., IN-1K/MOP1, IN-1K/MOP2, IN-1K/MOP4, IN-1K/MOP5, and IN-1K/MOP7), NSGA-II performs consistently better than other algorithms; by contrast, on problems with more than two objectives (i.e., IN-1K/MOP3, IN-1K/MOP8, and IN-1K/MOP9), IBEA performs the best among six algorithms. Additionally, without an explicit normalization mechanism, the decomposition-based algorithms (i.e., MOEA/D and RVEA) perform substantially worse than the others. For further analysis, we plot the nondominated solutions achieved by each algorithm in the median run on two typical test instances in Fig. 13a and Fig. 13b for IN-1K/MOP1 and IN-1K/MOP8 respectively. In the following, we provide some discussions based on the observations made from Fig. 13.

IN-1K/MOP1 is a bi-objective problem of simultaneous



(a) C-10/MOP1



(b) C-10/MOP6

Fig. 12: Nondominated solutions obtained by each algorithm on (a) C-10/MOP1 and (b) C-10/MOP6. We select the run associated with the median HV value. For each row, the subfigures correspond to NSGA-II, IBEA, MOEA/D, NSGA-III, HypE, and RVEA, respectively.

TABLE X: Statistical results (median and standard deviation) of the HV values on IN-1K/MOP test suite. The best results of each instance are in bold.

Problem	NSGA-II	IBEA	MOEA/D	NSGA-III	HypE	RVEA
IN-1K/MOP1	**0.9299 (0.0045)**$^+$	0.9246 (0.0054)$^-$	0.7447 (0.0494)$^-$	0.9196 (0.0124)$^-$	0.9020 (0.0075)$^-$	0.7662 (0.0372)$^-$
IN-1K/MOP2	**0.8849 (0.0011)**$^+$	0.8843 (0.0019)$^-$	0.5239 (0.0883)$^-$	0.8799 (0.0017)$^-$	0.8592 (0.0047)$^-$	0.2038 (0.1123)$^-$
IN-1K/MOP3	0.7945 (0.0051)$^-$	**0.8187 (0.0046)**$^+$	0.0603 (0.0579)$^-$	0.7881 (0.0624)$^-$	0.8079 (0.0046)$^-$	0.7382 (0.0178)$^-$
IN-1K/MOP4	**0.9930 (0.0048)**$^+$	0.9876 (0.0058)$^-$	0.6234 (0.0700)$^-$	0.9882 (0.0052)$^-$	0.9311 (0.0264)$^-$	0.3872 (0.1290)$^-$
IN-1K/MOP5	**0.9975 (0.0040)**$^+$	0.9913 (0.0060)$^-$	0.6143 (0.0833)$^-$	0.9908 (0.0049)$^-$	0.9311 (0.0301)$^-$	0.3416 (0.1351)$^-$
IN-1K/MOP6	**0.9832 (0.0041)**$^+$	0.9761 (0.0061)$^-$	0.1570 (0.0323)$^-$	0.9541 (0.0040)$^-$	0.8971 (0.0307)$^-$	0.8495 (0.0288)$^-$
IN-1K/MOP7	**0.9049 (0.0134)**$^+$	0.8895 (0.0197)$^-$	0.6324 (0.0707)$^-$	0.8843 (0.0164)$^-$	0.8687 (0.0156)$^-$	0.8119 (0.0280)$^-$
IN-1K/MOP8	0.6884 (0.0067)$^-$	**0.7267 (0.0054)**$^+$	0.0533 (0.1158)$^-$	0.6989 (0.0462)$^-$	0.7135 (0.0057)$^-$	0.5918 (0.0432)$^-$
IN-1K/MOP9	0.5783 (0.0000)$^-$	**0.6514 (0.0072)**$^+$	0.1416 (0.1002)$^-$	0.5956 (0.0060)$^-$	0.6269 (0.0122)$^-$	0.4946 (0.0394)$^-$

$^+$ indicates a method achieving significantly better performance.
$^-$ indicates a method achieving significantly worse performance.

minimization of prediction error (f^e) and No. of parameters (f_1^c). In addition to the multi-modal and noisy fitness landscape, another primary challenge of this problem is that the two objectives are of different scales: the objective of f^e falls into the range of $[0, 1]$, while the objective of f_1^c is in the range of millions. Consequently, as depicted in Fig. 13a, we can observe that both MOEA/D and RVEA fail to converge to the PF. By contrast, NSGA-III achieves comparable performance to indicator-based algorithms (i.e., IBEA and HypE) owing to its internal normalization mechanism. Similar observations can be made on IN-1K/MOP8 test instance from Fig. 13b.

D. Further Discussion

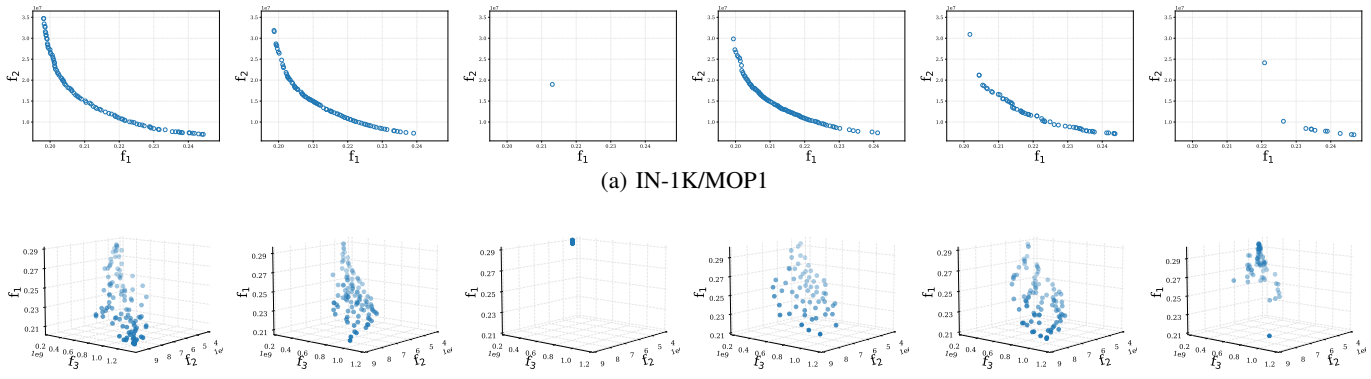
In addition to the primary goal of facilitating an empirical comparison among EMO algorithms, the results of these test suites also provide valuable insights toward understanding the effectiveness of DNNs from an architectural perspective. For instance, let us consider the C-10/MOP5, a five-objective problem for simultaneously optimizing prediction error (f^e), model complexity ($\boldsymbol{f}^c$) and hardware efficiency ($\boldsymbol{f}^{h_1}$) on GPUs. We visualize the Pareto architectures obtained by NSGA-II in the median run measured by the HV metric. More specifically, we consider architectures from three different trade-off subsets of the final nondominated solutions: (i) a subset preferred for f^e and $\boldsymbol{f}^c$; (ii) a subset preferred for f^e and $\boldsymbol{f}^{h_1}$; and (iii) a subset striking a balance among f^e, $\boldsymbol{f}^c$, and $\boldsymbol{f}^{h_1}$.

As depicted in Fig. 14d, there are consistent trends on the architectural choices (i.e., decision variables) across three different scenarios. In particular, we can observe that the 3×3 average pooling (i.e., "avg" in Fig. 14) is the least frequently used operator, while the 3×3 convolution (i.e., "3×3" in Fig. 14) is preferred consistently. Given that f^e is the common objective in all three scenarios, we argue that the 3×3 convolution is important for achieving a better prediction error, while the 3×3 average pooling adversely affects the prediction error. Moreover, we can observe that a transition from the f^e-$\boldsymbol{f}^c$ preferred subset to the f^e-$\boldsymbol{f}^{h_1}$ preferred subset can be achieved by gradually dropping the 1×1 convolution (i.e., replace "1×1" with "none" in Fig. 14a and 14b).

In the meantime, the results of these test suites also provide valuable feedback on hardware devices. For instance, as depicted in Fig. 12b, the inference latency, on one hand, is correlated (objective 4 in Fig. 12b) with the energy consumption (objective 5 in Fig. 12b) on Eyeriss[5], i.e., a DNN with a faster inference speed tends to draw less energy as well; on the other hand, however, trade-offs exist between the inference speed/energy consumption and the arithmetic intensity (objective 6 in Fig. 12b; defined as operations per unit memory traffic)[6].

As demonstrate above, running EMO algorithms on

[5]The lines between objective 4 and objective 5 are in parallel.
[6]The lines between objectives 5 and 6 are in a crisscross pattern.



(a) IN-1K/MOP1

(b) IN-1K/MOP8

Fig. 13: Nondominated solutions obtained by each algorithm on (a) IN-1K/MOP1 and (b) IN-1K/MOP8. We select the run associated with the median HV value. For each row, the subfigures correspond to NSGA-II, IBEA, MOEA/D, NSGA-III, HypE, and RVEA, respectively.

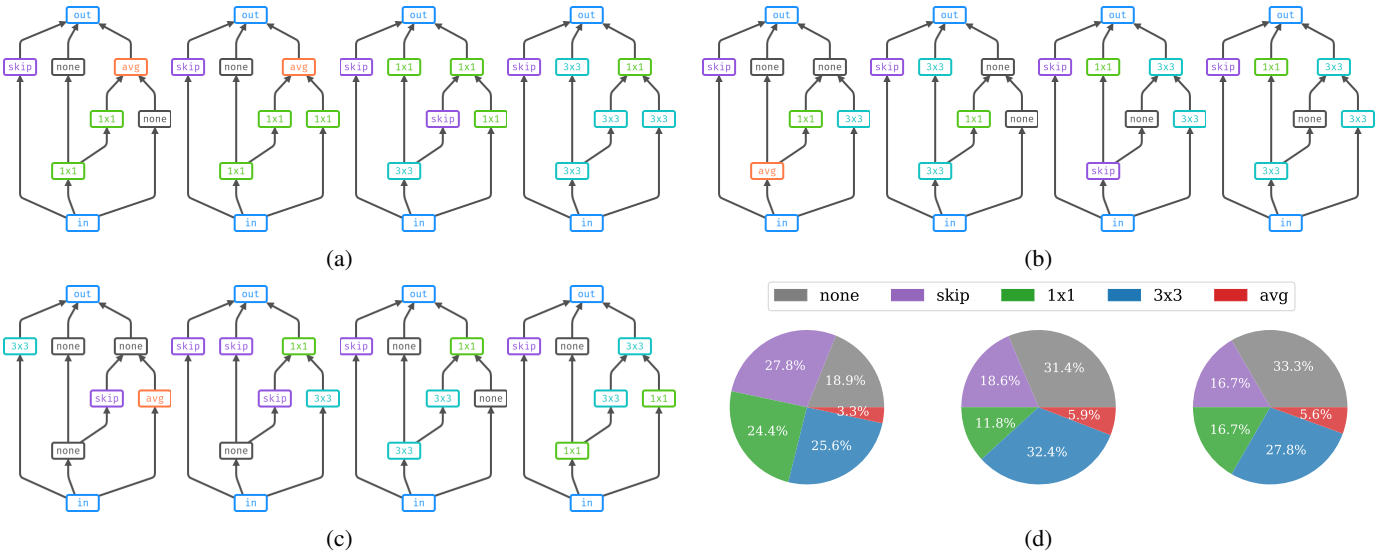


Fig. 14: The architectures obtained by NGSA-II on C-10/MOP5 in the run associated with the median HV value. We visualize architectures from three different trade-off subsets: (a) the subset preferred for prediction error and model complexity, i.e., nondominated considering prediction error (f^e), No. of parameters (f_1^c) and FLOPs (f_2^c); (b) the subset preferred for prediction error and hardware efficiency, i.e., nondominated considering prediction error (f^e), GPU latency ($f_1^{h_1}$) and energy ($f_2^{h_1}$); (c) the subset striking a balance between prediction error, model complexity, and hardware efficiency. (d) We also provide the frequency of the operators (i.e., decision variables of C-10/MOP5) used by these three subsets respectively (from left to right).

EvoXBench provides rich results worthy of deep investigations. Beyond the main target of NAS for automating hardware-related design/deployment of DNNs, the empirical observations can be also useful in hardware designs.

VII. Conclusion

This paper was devoted to bridging the gap between EMO and NAS. First, we provided a general multiobjective formulation of NAS together with the analyses of complex characteristics from the optimization point of view. Then, we presented an end-to-end pipeline – EvoXBench, to streamline the generation of benchmark test problems for EMO algorithms to run efficiently without the requirements of GPUs or sophisticated software such as Pytorch/TensorFlow. Next, we initiated two test suites comprising two image classification datasets, seven search spaces, three types of hardware, and up to eight objectives. Finally, an experimental study was conducted using six representative algorithms implemented in MATLAB, having demonstrated the effectiveness of EvoXBench and provided insightful analyses of the results.

On top of the generic problem formulations of NAS problems, EvoXBench provides a generic framework which can be extended to include additional search spaces, hardware devices, and tasks by following the pipeline as described in the paper. Meanwhile, more benchmark test instances can be also generated via EvoXBench.

ACKNOWLEDGEMENT

We wish to thank Beichen Huang for helping with the codes and experiments in MATLAB, Xukun Liu for helping with the database codes and monitoring the experiments, Yansong Huang for helping with the visualization, and Bowen Zheng for helping with the Transformer search space.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2012.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 27, 2014.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[7] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[8] Z. Lu, R. Cheng, S. Huang, H. Zhang, C. Qiu, and F. Yang, "Surrogate-assisted multiobjective neural architecture search for real-time semantic segmentation," *IEEE Transactions on Artificial Intelligence*, pp. 1–14, 2022.

[9] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. D. Goodman, W. Banzhaf, and V. N. Boddeti, "Multiobjective evolutionary design of deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 277–291, 2021.

[10] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 2971–2989, 2021.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[12] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007.

[13] J. Liang, X. Ban, K. Yu, B. Qu, K. Qiao, C. Yue, K. Chen, and K. C. Tan, "A survey on evolutionary constrained multi-objective optimization," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2022.

[14] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 2902–2911.

[15] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[16] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019.

[17] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4095–4104.

[18] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[19] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.

[20] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, "Platemo: A matlab platform for evolutionary multi-objective optimization [educational forum]," *IEEE Comput. Intell. Mag.*, vol. 12, no. 4, pp. 73–87, 2017.

[21] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 7105–7114.

[22] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[23] X. Dong, L. Liu, K. Musial, and B. Gabrys, "NATS-Bench: Benchmarking nas algorithms for architecture topology and size," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2021.

[24] A. Zela, J. N. Siems, L. Zimmer, J. Lukasik, M. Keuper, and F. Hutter, "Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[25] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[27] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," in *Evolutionary Methods for Design, Optimisation and Control*, 2002, pp. 95–100.

[28] D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates *et al.*, "PESA-II: Region-based selection in evolutionary multi-objective optimization," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*. ACM, 2001.

[29] H.-L. Liu, F. Gu, and Q. Zhang, "Decomposition of a multi-objective optimization problem into a number of simple multi-objective subproblems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 2450–455, 2014.

[30] E. Zitzler and S. Künzli, "Indicator-based selection in multi-objective search," in *Proceedings of the Parallel Problem Solving from Nature*. Springer, 2004, pp. 832–842.

[31] M. Emmerich, N. Beume, and B. Naujoks, "An emo algorithm using the hypervolume measure as selection criterion," in *Proceedings of the Evolutionary Multi-criterion Optimization*. Springer, 2005, pp. 62–76.

[32] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: empirical results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, 2000.

[33] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, *Evolutionary multi-objective optimization*. Springer, 2005, ch. Scalable test problems for evolutionary multiobjective optimization.

[34] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Trans. Evol. Comput.*, vol. 10, no. 5, pp. 477–506, 2006.

[35] R. Cheng, M. Li, Y. Tian, X. Zhang, S. Yang, Y. Jin, and X. Yao, "A benchmark test suite for evolutionary many-objective optimization," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 67–81, 2017.

[36] R. Cheng, Y. Jin, M. Olhofer *et al.*, "Test problems for large-scale multiobjective and many-objective optimization," *IEEE Trans. Cybern.*, vol. 47, no. 12, pp. 4108–4121, 2016.

[37] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[38] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 2820–2828.

[39] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[40] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, October 2019.

[41] K. T. Chitty-Venkata and A. K. Somani, "Neural architecture search survey: A hardware perspective," *ACM Computing Surveys (CSUR)*, 2022.

[42] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[44] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017.
[45] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.
[46] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
[47] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017.
[48] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS) 32*, 2019.
[49] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao, and Y. Lin, "HW-NAS-Bench: Hardware-aware neural architecture search benchmark," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.
[50] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, A. Filippov, and E. Burnaev, "Nas-bench-nlp: neural architecture search benchmark for natural language processing," *IEEE Access*, vol. 10, pp. 45 736–45 747, 2022.
[51] Y. Mehta, C. White, A. Zela, A. Krishnakumar, G. Zabergja, S. Moradian, M. Safari, K. Yu, and F. Hutter, "NAS-bench-suite: NAS evaluation is (now) surprisingly easy," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.
[52] Y. Duan, X. Chen, H. Xu, Z. Chen, X. Liang, T. Zhang, and Z. Li, "Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 5251–5260.
[53] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *J. Mach. Learn. Res.*, vol. 9, no. 86, pp. 2579–2605, 2008.
[54] P. Kerschke, H. Wang, M. Preuss, C. Grimme, A. H. Deutz, H. Trautmann, and M. T. Emmerich, "Search dynamics on multimodal multiobjective problems," *Evol. Comput.*, vol. 27, no. 4, pp. 577–609, 2019.
[55] R. Tanabe and H. Ishibuchi, "A review of evolutionary multimodal multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 193–200, 2019.
[56] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, 2005.
[57] P. Rakshit, A. Konar, and S. Das, "Noisy evolutionary optimization algorithms–a comprehensive survey," *Swarm and Evol. Comput.*, vol. 33, pp. 18–45, 2017.
[58] C. K. Goh and K. C. Tan, "An investigation on noisy environments in evolutionary multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 11, no. 3, pp. 354–381, 2007.
[59] J. E. Fieldsend and R. M. Everson, "The rolling tide evolutionary algorithm: A multiobjective optimizer for noisy optimization problems," *IEEE Trans. Evol. Comput.*, vol. 19, no. 1, pp. 103–117, 2015.
[60] B. Li, J. Li, K. Tang, and X. Yao, "Many-objective evolutionary algorithms: A survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–35, 2015.
[61] L. He, H. Ishibuchi, A. Trivedi, H. Wang, Y. Nan, and D. Srinivasan, "A survey of normalization methods in multiobjective evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 25, no. 6, pp. 1028–1048, 2021.
[62] H. Ishibuchi, H. Masuda, and Y. Nojima, "Pareto fronts of many-objective degenerate test problems," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 807–813, 2015.
[63] M. Elarbi, S. Bechikh, C. A. C. Coello, M. Makhlouf, and L. B. Said, "Approximating complex pareto fronts with predefined normal-boundary intersection directions," *IEEE Trans. Evol. Comput.*, vol. 24, no. 5, pp. 809–823, 2019.
[64] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.
[65] M. Chen, H. Peng, J. Fu, and H. Ling, "AutoFormer: Searching transformers for visual recognition," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 12 270–12 280.
[66] Y. Xu, Y. Wang, K. Han, Y. Tang, S. Jui, C. Xu, and C. Xu, "Renas: Relativistic evaluation of neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 4411–4420.
[67] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda *et al.*, "Fbnetv3: Joint architecture-recipe search using predictor pretraining," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 16 276–16 285.
[68] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.
[69] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, 2014.
[70] J. Bader and E. Zitzler, "HypE: an algorithm for fast hypervolume-based many-objective optimization," *Evol. Comput.*, vol. 19, no. 1, pp. 45–76, 2011.
[71] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "A reference vector guided evolutionary algorithm for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 773–791, 2016.
[72] C. M. Fonseca, P. J. Fleming *et al.*, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proceedings of the International Conference on Genetic Algorithms*, vol. 93, 1993, pp. 416–423.
[73] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, 1994.
[74] L. Xie and A. Yuille, "Genetic cnn," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 1379–1388.
[75] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2017, pp. 497–504.
[76] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2019.
[77] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, 2020.
[78] B. Baker*, O. Gupta*, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in *Proc. Int. Conf. Learn. Represent. (ICLR) Workshop*, 2018.
[79] H. Zhang, Y. Jin, Y. Jin, and K. Hao, "Evolutionary search for complete neural network architectures with partial weight sharing," *IEEE Trans. Evol. Comput.*, pp. 1–1, 2022.
[80] H. Tan, R. Cheng, S. Huang, C. He, C. Qiu, F. Yang, and P. Luo, "Relativenas: Relative neural architecture search via slow-fast learning," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–15, 2021.
[81] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.
[82] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, 2020.
[83] S. Liu, H. Zhang, and Y. Jin, "A survey on surrogate-assisted efficient neural architecture search," *arXiv preprint arXiv:2206.01520*, 2022.
[84] I. Das and J. E. Dennis, "Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems," *SIAM journal on optimization*, vol. 8, no. 3, pp. 631–657, 1998.

APPENDIX

A. Extended Description of Evolutionary Multiobjective Optimization

Dating back to the 1990s, the *dominance-based* EMO algorithms were originally motivated to tailor the selection operators of EAs on the basis of *Pareto dominance* [72], [73]. Later, it was proposed that the use of *elitism mechanism* would substantially improve the convergence of the algorithms. Afterward, the emergence of several milestone algorithms such as the elitist non-dominated sorting genetic algorithm (NSGA-II) [26], the strength Pareto evolutionary

algorithms (SPEA2) [27] and the Pareto envelope-based selection algorithm (PESA) [28] has laid a solid foundation for future research of the field.

Following the classic divide-and-conquer strategies as widely adopted in algorithm designs, more recently, some *decomposition-based* approaches have been proposed to divide a complex MOP into a number of sub-problems (either multiobjective or single objective ones) and conquer them in a collaborative manner. As the milestone EMO algorithm of this category, the multiobjective evolutionary algorithm based on decomposition (MOEA/D) [12] was proposed to divide an MOP into a number of scalar optimization sub-problems via weighted aggregation of the multiple objectives. Another representative algorithm of this category is the MOEA/D-M2M [29], which was proposed to divide an MOP into a number of simple multiobjective sub-problems via objective space division.

Since the performance of an EMO algorithm is measured by the performance indicators, intuitively, various *indicator-based* EMO algorithms have been proposed. An early milestone is the binary-indicator-($I_{\epsilon+}$)-based evolutionary algorithm (IBEA) [30], where $I_{\epsilon+}$ is considered to be compliant with the Pareto dominance. Following this pathway, some works proposed to integrate EMO algorithms with a more powerful performance indicator – the hypervolume indicator (or the S metric), e.g., the s-metric selection evolutionary multiobjective algorithm (SMS-EMOA) [31].

B. Literature Review of NAS Continued

1) A Brief Review of Existing NAS Search Algorithms: In general, existing NAS approaches can be categorized into reinforcement learning (RL) based ones, differentiable/gradient-based ones (DARTS), and evolutionary algorithm (EA) based ones. RL treats the design of network architecture as a sequential decision process, where an agent is trained to optimally choose pieces to assemble the network [7]. Despite that modeling NAS into an RL task is intuitive, RL-based approaches often require a large number of iterations (a.k.a *episodes*) to converge, resulting in weeks of wall clock time on hundreds of GPUs. DARTS seeks to improve search efficiency by problem reformulation via continuous relaxation, allowing the architectural parameters (i.e., the decision variables of a NAS problem) to be jointly optimized with the weights by gradient descent through back-propagation [18]. However, such a problem reformulation suffers from excessive GPU memory requirements during the search, resulting in overly confined search spaces such as reduced layer operation choices. EA treats NAS as a black-box discrete optimization problem, where a population of candidate solutions is iterated to be gradually better via genetic operations or heuristics [74]. Due to its modular framework, flexible encoding, and population-based nature, EA has attracted increasing attention, leading to a plethora of emerging EA-based NAS approaches [14], [75]–[77].

Despite that, there are advanced methods such as weight sharing [17] to improve RL's search efficiency, or the binary gating [39] to reduce the GPU memory footprint of DARTS, most existing non-EA-based methods are not directly applicable to multiobjective NAS. To this end, this work is dedicated to standardizing the problem of NAS in terms of both formulation and benchmarking, such that previous algorithmic efforts from the EMO community can be inherited and future algorithm designs can be properly and fairly assessed.

Algorithm 1: Pseudocode of a search space class

```
class SearchSpace():
    def __init__(self, **kwargs):
        arch: phenotype # used for querying fitness
        x: genotype # used by genetic operators

    def is_valid(arch):
        # a method to verify the validity of a solution
        return True/False

    def sample():
        # a method to randomly create a solution
        return x

    def encode(arch):
        # a method to convert phenotype to genotype
        return x

    def decode(x):
        # a method to convert genotype to phenotype
        return arch
```

2) A Brief Review of Existing NAS Fitness Evaluators: A thorough training from scratch to assess the prediction error (i.e., fitness) of an architecture is computationally prohibitive for realistically-sized search spaces [15], [18], [38]. Consequently, architectures are almost always evaluated *imperfectly* during a NAS process. Specifically, architectures are typically evaluated with the following methods: (i) proxy tasks (e.g., small-scale architectures [9] and fewer training epochs [78]); (ii) surrogate models (e.g., inherited weights from a supernet [79], [80] or a "neighbor" architecture (network morphism) [81], performance predictors [82], [83]); (iii) a fixed set of training hyperparameters, neglecting the fact that different architectures are likely to enjoy different sets of hyperparameters to be optimal in training [67].

All search spaces in `EvoXBench` follow the canonical schema outlined in Algorithm 1. Specifically, a search space object facilitates the following four functions:

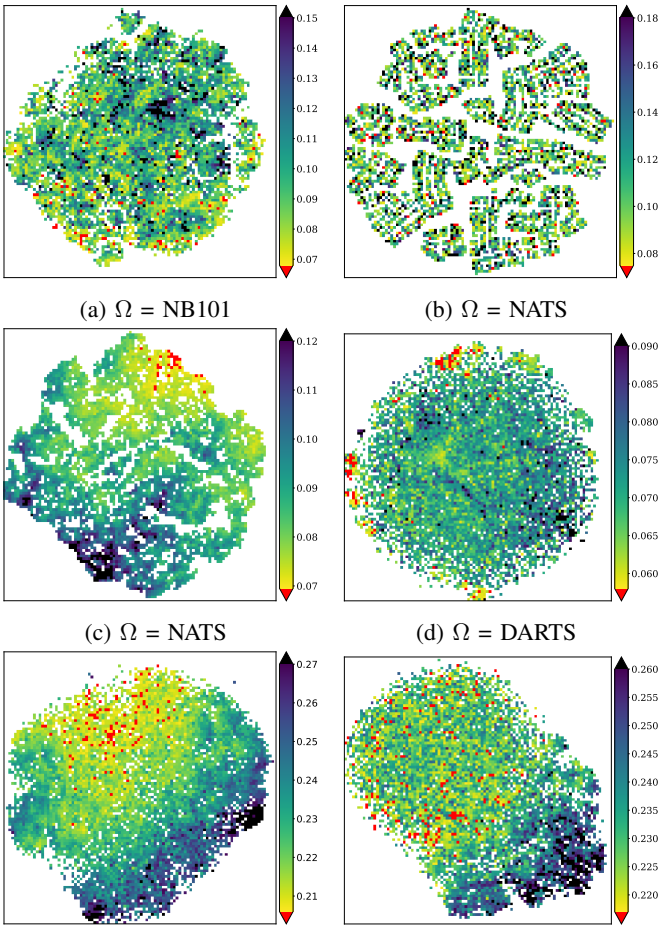
1) `is_valid`: a method to verify the validity of an architecture phenotype (`arch`) before an architecture can be sent for fitness evaluation.
2) `sample`: a method to randomly create a *valid* architecture genotype `x`. Note that, for some search spaces such as NB101, the default sampling methods (e.g., uniform or Latin hypercube) in existing EMO algorithms do not always produce a valid solution.
3) `encode`: a method to convert an architecture phenotype (`arch`) to its corresponding genotype (`x`). Note that this is *NOT* a one-to-one mapping, i.e., there exist many genotypes that correspond to one (the same) phenotype.
4) `decode`: a method to convert an architecture genotype (`x`) to its corresponding phenotype (`arch`).

The default `encode` method in `EvoXBench` is a simple concatenation of integer values to indicate the choices of architectural elements. We acknowledge that the `encode`

method itself is an important design choice for NAS and can (or rather should) be studied/explored further. However, such studies are beyond the scope of this paper.

C. Fitness Landscape

In order to visualize the fitness landscapes of the seven search spaces shown in Table II, we need to project the high-dimensional decision variables to a two-dimensional space. We consider two projection methods, i.e., principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) [53]. And we use 10,000 solutions uniformly sampled from each search space. The visualizations are provided in Fig. A1 and Fig. A2 respectively for t-SNE and PCA.



(a) Ω = NB101 (b) Ω = NATS

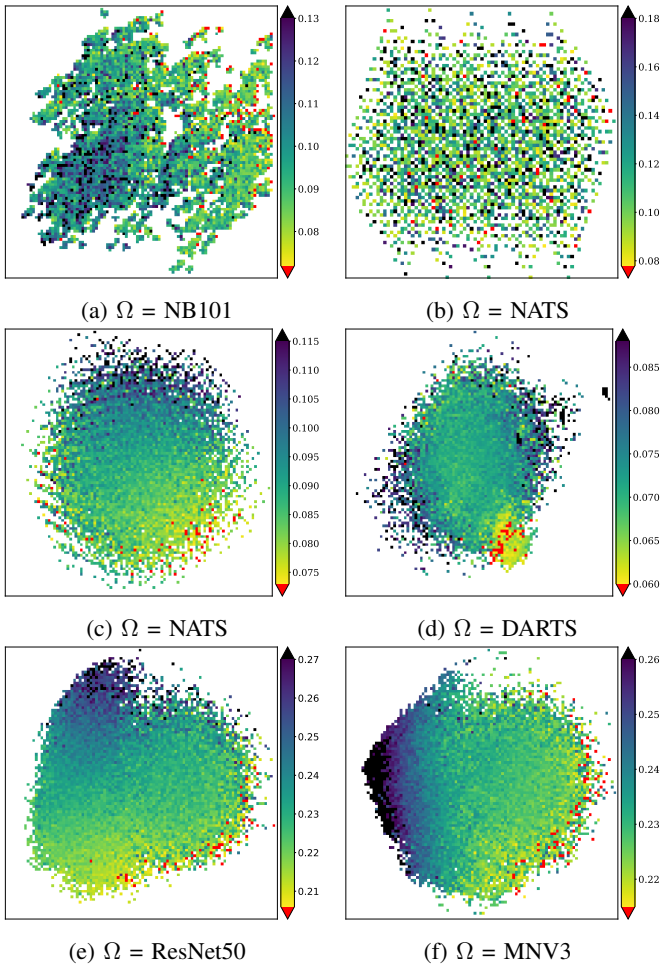
(c) Ω = NATS (d) Ω = DARTS

(e) Ω = ResNet50 (f) Ω = MNV3

Fig. A1: Fitness landscapes of f^e (i.e. prediction error) on (a) NB201, (b) NATS, (c) ResNet50, and (d) MNV3. We project the original high-dimensional decision space to 2D latent space via t-SNE [53]. We random sample 10K solutions from each search space and average f^e within each small area. The top-performing solutions are highlighted in red.

In general, the nature of these two objectives (i.e., number of parameters and FLOPs) depends on the search space and the architecture. Hence, whether these two objectives are correlated or in conflict varies from case to case. We have visualized the correlation between these two objectives (i.e., number of parameters and FLOPs) for architectures from the search spaces supported in `EvoXBench` in Fig. A3. Evidently, we observe that two clusters of search spaces emerge:

- For one cluster of search spaces (i.e., NB101, NB201, and DARTS), their architecture decision variables mainly represent topological choices and connections, i.e., which operator (e.g., convolution or pooling) to use and how these operators are connected (see Figure 3 in the main paper for visualizations). We observe that these two objectives (i.e., number of parameters and FLOPs) are in harmony for architectures sampled from this cluster of search spaces.
- For the other cluster of search spaces (i.e., NATS, MobileNetV3, and ResNet50), their architecture decision variables mainly represent the size and hyperparameters of the network (i.e., number of layers, channels, and kernel sizes, etc.). We observe that these two objectives (i.e., number of parameters and FLOPs) are in conflict for architectures sampled from this cluster of search spaces.

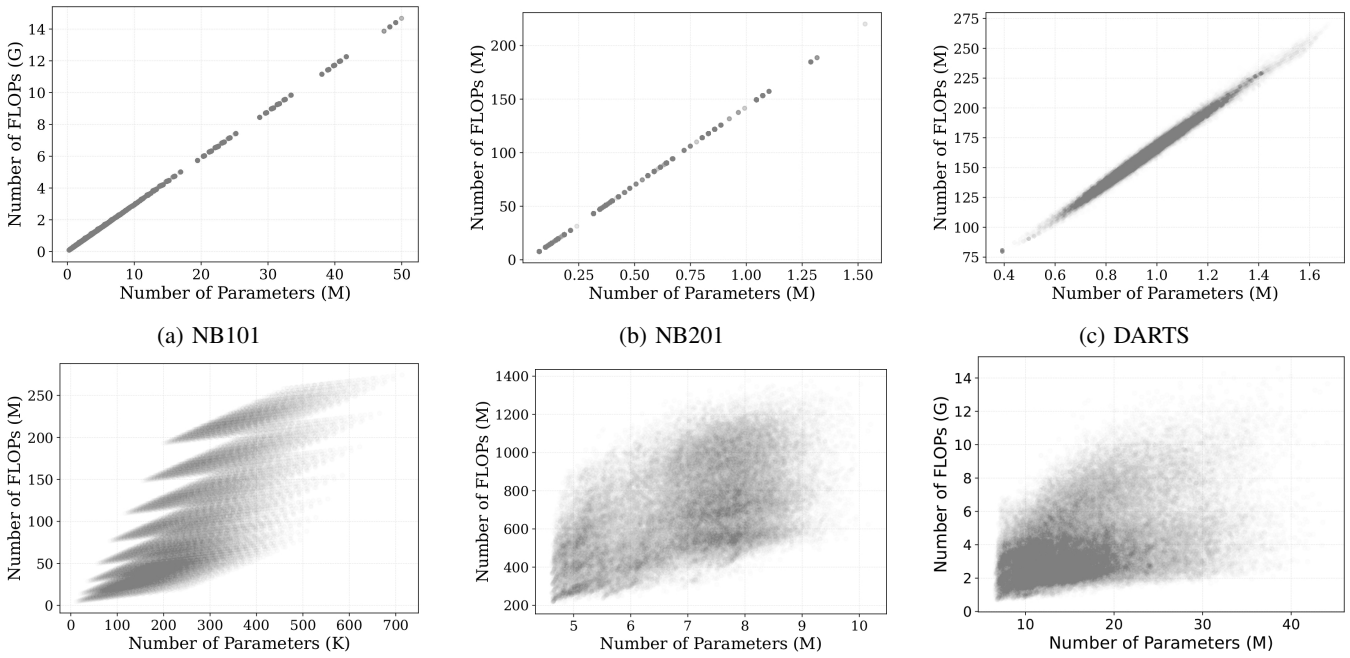


(a) Ω = NB101 (b) Ω = NATS

(c) Ω = NATS (d) Ω = DARTS

(e) Ω = ResNet50 (f) Ω = MNV3

Fig. A2: Fitness landscapes of f^e (i.e. prediction error) on (a) NB201, (b) NATS, (c) ResNet50, and (d) MNV3. We project the original high-dimensional decision space to 2D latent space via PCA. We random sample 10K solutions from each search space and average f^e within each small area. The top-performing solutions are highlighted in red.



(a) NB101 (b) NB201 (c) DARTS

(d) NATS (e) MobileNetV3 (f) ResNet50

Fig. A3: Visualization of correlation between number of parameters and FLOPs of the architectures from the search spaces supported in `EvoXBench`.

Further investigations are required to dissect the fundamental cause of these diverging behaviors from the two clusters of search spaces. However, such investigations are beyond the scope of this work. Nevertheless, we believe that these interesting observations can be viewed as additional evidences towards supporting the claim that our `EvoXBench` include a diverse set of search spaces of NAS.

D. Implementation Details

1) Settings of population size: We follow the Das-and-Dennis' approach to generate reference directions [84] for MOEA/D, NSGA-III, and RVEA. The population size N is then determined by the simplex-lattice factor H and the number of objectives M. For test instances with more than five objectives, we follow the bi-layer reference direction generation method as recommended in [69]. The detailed settings are summarized in Table V.III in the main paper. And to be consistent, we use the same population size for NSGA-II, IBEA, and HypE. Note that we also round N to be a multiple of four and two for NSGA-II and NSGA-III, respectively.

2) Settings of termination: A maximum of 10,000 function evaluations is used as the termination criterion for all algorithms. We perform 31 repetitions to each algorithm for a statistically meaningful comparison.

3) Other settings: we follow the default settings provided by the original papers to configure the genetic operators and the hyperparameters of the algorithms. And the settings of reference points for calculating the HV metric are provided in Table A.I. Additionally, we visualize the true PFs in Fig. A3 for C-10/MOP1 - C-10/MOP7, where all attainable solutions are exhaustively evaluated.

E. IGD results

We present the statistical values of the IGD metric achieved by various EMO algorithms in Table A.II. Note that we only consider the first seven instances from the C-10/MOP test suite, whose true PFs are known from exhaustive evaluations.

– By a new search space: depending on the volume, there are two ways to incorporate a new search space into `EvoXBench`. The specific procedures are outlined below with a pictorial illustration shown in Figure A4 and an overview comparison provided in Table A.III.

1) If the volume of the new search space is confined such that an enumeration is possible (e.g., typically the total number of unique solutions/architectures should be $< 10^5$), we will simply exhaustively evaluate them all. Specifically, we train all solutions thoroughly from scratch on the training set and then evaluate them on the validation set. We repeat this process three times and store the results in a database. See the top path shaded in blue in Figure A4.
2) If the volume of the new search space is beyond enumeration, we resort to surrogate modeling. First, we train a supernet – a composite network such that every attainable architecture (solution) becomes a sub-part of it. Second, we validate the effectiveness of the trained supernet on a subset of architectures uniformly sampled from the search space (we sample 100 architectures in this work). Depending on if the performance (i.e., prediction error, f^e) calculated based on the inherited weights (from the supernet) is sufficiently correlated with the performance calculated based on the thoroughly trained from-scratch weights, there are two ways to

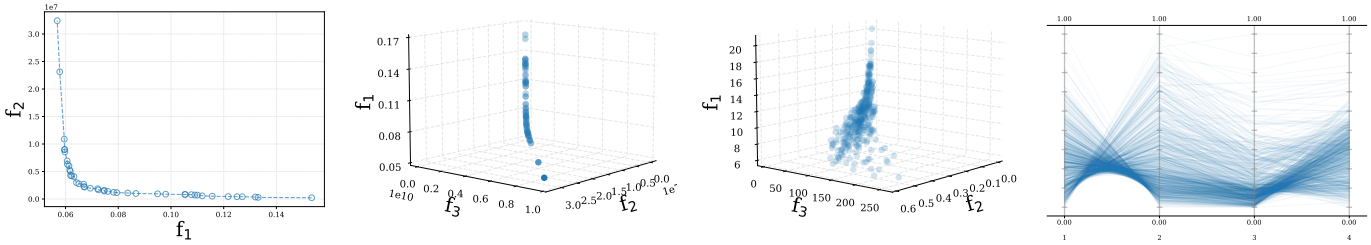
Table A.I: The reference points used for calculating the HV metric on (a) C-10/MOP and (b) IN-1K/MOP test suites.

(a) C-10/MOP

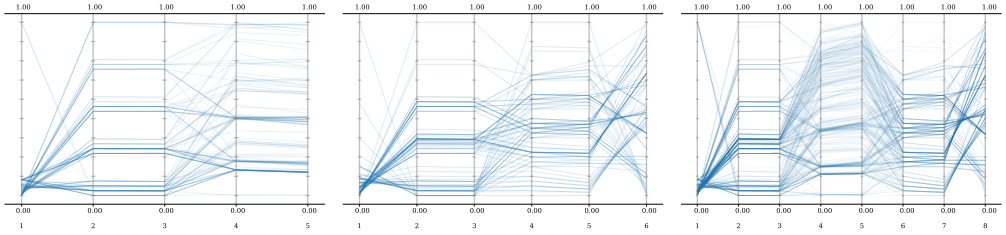
Problem	Reference point
C-10/MOP1	[0.1534, 3.2427e7]
C-10/MOP2	[0.1577, 3.2427e7, 9.5450e9]
C-10/MOP3	[0.2021, 5.7995e5, 2.5706e8]
C-10/MOP4	[0.2021, 5.7995e5, 2.5706e8, 2.0064e-2]
C-10/MOP5	[0.9000, 1.0735e6, 1.5327e8, 6.8889e-3, 3.2651e-2]
C-10/MOP6	[0.5098, 1.0735e6, 1.5327e8, 1.0527e-2, 2.0139e-3, 26.596]
C-10/MOP7	[0.9000, 1.0735e6, 1.5327e8, 8.1821e-3, 3.4711e-2, 1.0527e-2, 2.0139e-3, 27.078]
C-10/MOP8	[0.2750, 1.6724e6]
C-10/MOP9	[0.2750, 1.6724e6, 2.7034e8]

(b) IN-1K/MOP

Problem	Reference point
IN-1K/MOP1	[0.3124, 4.4114e7]
IN-1K/MOP2	[0.3124, 1.4577e10]
IN-1K/MOP3	[0.3124, 4.4114e7, 1.4577e10]
IN-1K/MOP4	[0.1832, 7.4134e7]
IN-1K/MOP5	[0.1832, 1.5403e10]
IN-1K/MOP6	[0.1832, 7.4134e7, 1.5403e10]
IN-1K/MOP7	[0.2980, 1.0198e7]
IN-1K/MOP8	[0.2980, 1.0198e7, 1.3768e9]
IN-1K/MOP9	[0.2980, 1.0198e7, 1.3768e9, 7.0386e-2]



(a) C-10/MOP1 (b) C-10/MOP2 (c) C-10/MOP3 (d) C-10/MOP4



(e) C-10/MOP5 (f) C-10/MOP6 (g) C-10/MOP7

Fig. A4: The true PFs of C-10/MOP1 - C-10/MOP7.

TABLE A.II: Statistical results (median and standard deviation) of the IGD values on the first seven instances of C-10/MOP test suite. The best results of each instance are in bold.

EMO Algorithms	C-10/MOP1	C-10/MOP2	C-10/MOP3	C-10/MOP4	C-10/MOP5	C-10/MOP6	C-10/MOP7
NSGA-II	**0.0309 (0.011)**	**0.0278 (0.009)**	**0.0235 (0.001)**	**0.0591 (0.002)**	**0.0054 (0.001)**	**0.0003 (0.000)**	**0.0444 (0.003)**
IBEA	0.0917 (0.014)	0.0927 (0.020)	0.0248 (0.002)	0.0736 (0.005)	0.2030 (0.044)	0.4025 (0.025)	0.5005 (0.017)
MOEA/D	0.0764 (0.015)	0.2239 (0.014)	0.0802 (0.010)	0.1309 (0.009)	0.5590 (0.016)	0.5607 (0.048)	0.7520 (0.078)
NSGA-III	0.0346 (0.009)	0.3437 (0.294)	0.0471 (0.003)	0.0933 (0.005)	0.3901 (0.006)	0.4569 (0.014)	0.6061 (0.038)
HypE	0.1194 (0.047)	0.1317 (0.054)	0.0486 (0.004)	0.0765 (0.004)	0.1021 (0.022)	0.0699 (0.016)	0.1713 (0.017)
RVEA*	0.0505 (0.011)	0.0585 (0.007)	0.0443 (0.005)	0.0849 (0.008)	0.1937 (0.030)	0.1421 (0.025)	0.2736 (0.037)

proceed:

- If the Kendall τ rank correlation is greater than the threshold (θ which is set to 0.7 in this work), the surrogate models are learned based on the objective values derived from the supernet. See the middle path shaded in green in Figure A4.
- Otherwise, we discard the supernet and revert to training from scratch to obtain the objective values for learning the surrogate models. See the bottom path shaded in red in Figure A4.

– By a new task: more complex vision tasks (e.g., semantic segmentation) can be naturally incorporated into `EvoXBench` following the same steps of adding a new search space, as shown above.

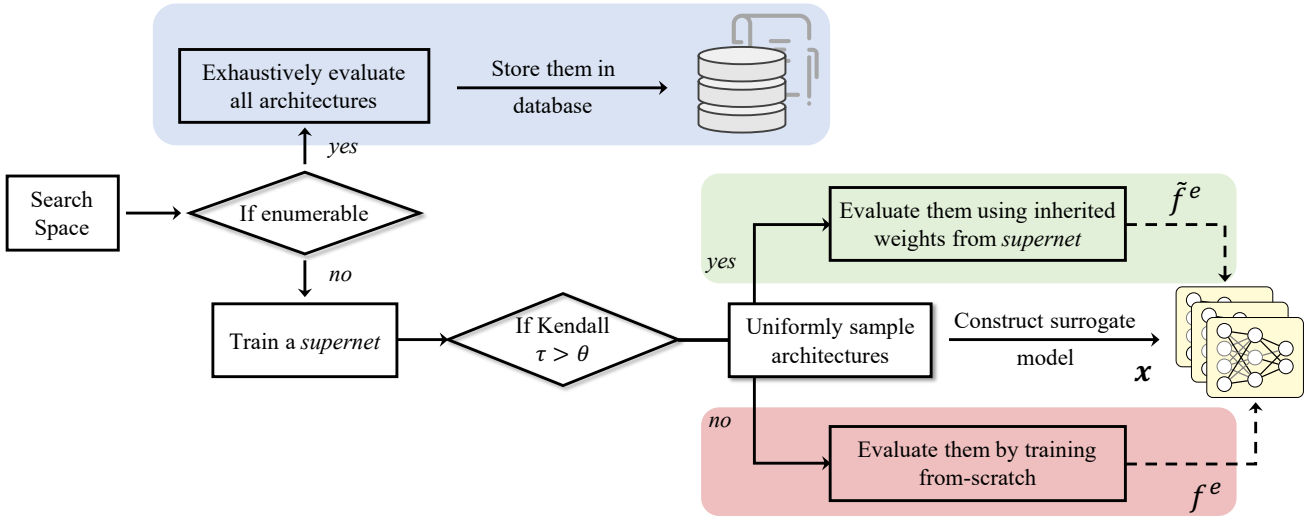


Fig. A4: The procedure flowchart for adding a new search space to `EvoXBench`.

TABLE A.III: An overview comparison among the three ways of incorporating a search space to `EvoXBench` shown in Figure A4.

Search spaces derived from	Pros	Cons	Examples
Exhaustive evaluation	• Actual objective landscapes are completely known.	• Typically confined in size. • Expensive to construct.	NB101, NB201, NATS
Surrogate modeling with supernet	• Efficient to construct. • Mild restriction on search space size.	• The supernet training is not trivial. • Objective landscapes are approximated.	ResNet50, Transformer, MNV3
Surrogate modeling without supernet	• Support almost any search space.	• Expensive to construct. • Objective landscapes are approximated. • Final obtained architecture (solution) needs to be re-trained.	DARTS