BenchENAS: A Benchmarking Platform for Evolutionary Neural Architecture Search

Xiangning Xie, Yuqiao Liu, *Graduate Student Member, IEEE*, Yanan Sun, *Member, IEEE*, Gary G. Yen, *Fellow, IEEE*, Bing Xue, *Senior Member, IEEE*, and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Neural architecture search (NAS), which automatically designs the architectures of deep neural networks, has achieved breakthrough success over many applications in the past few years. Among different classes of NAS methods, evolutionary computation-based NAS (ENAS) methods have recently gained much attention. Unfortunately, the development of ENAS is hindered by unfair comparison between different ENAS algorithms due to different training conditions and high computational cost caused by expensive performance evaluation. This article develops a platform named BenchENAS, in short for benchmarking evolutionary NAS, to address these issues. BenchENAS makes it easy to achieve fair comparisons between different algorithms by keeping them under the same settings. To accelerate the performance evaluation in a common lab environment, BenchENAS designs a novel and generic efficient evaluation method for the population characteristics of evolutionary computation. This method has greatly improved the efficiency of the evaluation. Furthermore, BenchENAS is easy to install and highly configurable and modular, which brings benefits in good usability and easy extensibility. This article conducts efficient comparison experiments on eight ENAS algorithms with high GPU utilization on this platform. The experiments validate that the fair comparison issue does exist in the current ENAS algorithms, and BenchENAS can alleviate this issue. A Website has been built to promote BenchENAS at https://benchenas.com, where interested researchers can obtain the source code and document of BenchENAS for free.**

***Index Terms*—Benchmarking platform, evolutionary computation (EC), neural architecture search (NAS).**

I. Introduction

Deep neural networks (DNNs), as the cornerstone of deep learning techniques [1], have achieved remarkable success in many real-world applications, such as classifying objects from images [2], [3], reasoning natural languages from texts [4], and recognizing speech from voice signals [5], to name a few. The vast successes of DNNs are generally credited due to the design of novel DNN architectures. This can be evidenced from VGG [6], ResNet [2], DenseNet [3], and Transformer [7], which have significantly different neural architectures from each other. Commonly, such novel architectures of DNNs are often manually designed with rich expertise [2], [3], [6]. However, with the increasing number of the state-of-the-art DNN building blocks, such as inception modules [8], residual connections [2], or dense connections [3], integrated into a high-performance DNN architecture, it is increasingly difficult to handcraft DNN architectures with exceptional performance. Furthermore, with the remarkable performance of DNNs, more and more researchers who are not experts of DNNs are trying to explore the amazing functionality of DNNs for the task at hand. Unfortunately, due to their limit or even without knowledge in designing promising DNN architectures, it is hard to obtain DNNs with satisfactory performance. This urgent demand has promoted the feverish development of neural architecture search (NAS) techniques [9], which aims to automatically generate robust and well-performing DNN architectures by formulating as optimization problems and then solved via well-designed optimization algorithms.

Based on the optimization algorithms adopted, existing NAS algorithms can be broadly classified into three different categories: 1) reinforcement learning (RL) [10]-based NAS algorithms [9]; 2) gradient-based NAS algorithms [11]; and 3) evolutionary computation (EC) [12]-based NAS algorithms (in short named ENAS) [13]. Specifically, the RL-based NAS algorithms often consume heavy computational resources due largely to the inexact reward resulted from RL techniques [14]. The gradient-based algorithms are more efficient than the RL-based algorithms. Unfortunately, the gradient-based algorithms require constructing a supernet in advance, which also demands specialized expertise. The ENAS algorithms solve the NAS problems by exploiting EC techniques. Specifically, EC is a class of computational paradigms, including genetic algorithms (GAs) [15], genetic programming (GP) [16], differential evolution (DE) [17], and particle swarm optimization (PSO) [18] solving challenging optimization problems by simulating the evolution of biology or swarming social behavior [19]–[23]. In EC, an initial population of candidate solutions is generated and iteratively updated by the evolutionary operators for generations. Each individual in each

Manuscript received 14 August 2021; revised 22 November 2021; accepted 26 January 2022. Date of publication 31 January 2022; date of current version 1 December 2022. This work was supported in part by Zhejiang Lab under Grant 2022PG0AB02; in part by China Computer Federation (CCF)-Baidu Open Fund under Grant 2021PP15002000; in part by Chinese Association for Artificial Intelligence (CAAI)-Huawei MindSpore Open Fund; and in part by the Science and Technology Innovation Talent Project of Sichuan Province under Grant 2021JDRC0001. *(Corresponding author: Yanan Sun.)*

Xiangning Xie, Yuqiao Liu, and Yanan Sun are with the College of Computer Science, Sichuan University, Chengdu 610065, China (e-mail: xiangningxie99@gmail.com; lyqguitar@gmail.com; ysun@scu.edu.cn).

Gary G. Yen is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078 USA (e-mail: gyen@okstate.edu).

Bing Xue and Mengjie Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: bing.xue@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2022.3147526.

Digital Object Identifier 10.1109/TEVC.2022.3147526

generation needs to be evaluated to obtain the fitness value. As a result, the population will gradually evolve to increase in fitness to arrive at the best solution. Compared to the RL-based NAS algorithms, ENAS algorithms often require less computation budget. Compared to the gradient-based NAS algorithms, ENAS algorithms are often fully automatic, and they can produce an appropriate NAS without any human intervention [14], [24]. ENAS algorithms have accounted for the majority of existing NAS algorithms as evidenced from a recent survey paper [25].

During past years, ENAS algorithms have attracted great attention owing to their high robustness, exceptional performance, and full ability to automate DNN architectures design [26], [27]. For example, the LargeEvo algorithm [13] first used GA to automate the discovery of DNN architectures for image classification. Meanwhile, the Genetic-CNN algorithm [28] proposed a fixed-length binary-string encoding method within GA to represent each network architecture. Furthermore, the Hierarchical-Evo algorithm [29] combined a new hierarchical genetic representation scheme to achieve efficient NAS. The EvoCNN algorithm [14] first developed the variable-length encoding strategy to search for promising DNN architectures without requiring manual effort during the search stage. The CGP-CNN algorithm [30] automatically constructed CNN architectures for image classification based on Cartesian GP [31], which is a type of GP [31]. In addition, the AE-CNN algorithm [24] and the CNN-GA algorithm [32] proposed to automatically evolve CNN architectures by using GA based on state-of-the-art CNN blocks [2], [3]. The NSGA-Net [33] algorithm achieved NAS by considering multiple conflicting targets using multiobjective EC algorithms [34]. The Regularized-Evo algorithm [35] introduced age attributes in a modification of tournament selection of GA, evolving an image classifier surpassing manual design for the first time.

Although ENAS researchers have drawn more attention to the community, there are still two challenges that need to be addressed. Specifically, the two challenges are the fair comparison issue and the efficient evaluation issue, as will be detailed as follows.

The fair comparison issue is widely recognized in ENAS. Making fair comparisons between different ENAS algorithms for ENAS developers is challenging, if not impossible. This has somehow hindered the development of ENAS because unfair comparisons may mislead the researchers and result in frustration. In addition, the researchers cannot exactly know how novel or competitive their algorithms are. Since ENAS is widely used in image classification tasks, taking the image classification task as an example, the classification accuracy (or classification error rate) and the computation budget of the searched architectures are the two most popular indicators used to evaluate the performance of existing ENAS algorithms [36]. When an ENAS algorithm is proposed and its performance is investigated on the classification accuracy, the developer needs to collect the classification accuracy values from some chosen state-of-the-art ENAS algorithms. However, because most ENAS algorithms are not open source, reproducing them under the same condition for arriving at fair classification accuracy is a nontrivial matter [37], [38]. In the most common practice, the reported classification accuracies of peer competitors for comparison are often extracted directly from their respective seminal papers. However, the experimental setups of these peer competitors are vastly different from each other, such as using different data preprocessing [39]–[41], different optimizers [42], [43], different learning rates, different batch sizes, and different training epochs, which are all deciding factors of the classification accuracy resulted by these ENAS algorithms. In this regard, the comparisons in terms of classification accuracy are clearly biased. On the other hand, the common practice for comparing with computation budget of ENAS algorithms is to measure by "GPU days" (i.e., the number of GPUs used $\times$ the days elapsed). Due to the different types of GPUs used by different ENAS developers, it is difficult to directly compare the computation budget of different ENAS algorithms by directly citing from their seminal papers. As a result, the comparison in terms of the collected "GPU days" is also unfair. On the other hand, the number of function evaluations (i.e., the generation number $\times$ the population size) is a key metric to fairly compare the performance of EC methods. Based on the extensive observations from the seminal papers of existing ENAS algorithms, most of them used different population sizes and generation numbers that result in different function evaluation numbers, and this is even worse as the individuals in the population(s) are of variable length instead of fixed length. This again leads to unfair comparisons among ENAS algorithms.

In addition, the lack of efficient evaluation methods has also hindered the development of ENAS. The ENAS algorithms depend upon the heavy requirement of computational resources because the fitness evaluation of DNNs during the evolutionary search will consume a lot of time. Specifically, the fitness evaluation of a DNN is achieved by training the DNN on the target dataset via a training-from-scratch process, which is nevertheless time consuming. For example, training a DNN on a common dataset such as CIFAR-10 [44] often consumes hours to days depending on the scale of the DNN. Moreover, since the EC methods are population based [16], [45], [46], there are many individual DNNs to be evaluated during the search process of the ENAS, which greatly increases the computational overhead of ENAS. For a common research environment, there are often multiple GPUs available in the lab. As a result, in-lab researchers often pursuit an acceleration of running the ENAS algorithm by evaluating their fitness on multiple GPUs. There are usually two strategies for using multiple GPUs. One strategy is evaluating the fitness of individuals with the in-house distributed parallel methods from the deep learning libraries, such as PyTorch [47] and Tensorflow [48], and the other strategy is to use distributed NAS toolkits. For the first strategy, these in-house distributed parallel methods use only multiple GPUs for the evaluation of a single DNN, which is not suitable for evaluating a large number of DNNs in ENAS. In this method, different GPUs need to communicate with each other to transfer computational parameters, which unavoidably increases the communication overhead and inadvertently increases the time for individual DNN training. The second strategy is a more effective approach to speeding up the ENAS algorithm

TABLE I
NINE ENAS ALGORITHMS INCLUDED IN THE VERSION 1.0 OF BENCHENAS

Algorithm	**Year**	**Paper**
LargeEvo [13]	2017	Large-Scale Evolution of Image Classifiers
CGP-CNN [30]	2017	A Genetic Programming Approach to Designing Convolutional Neural Network Architecture
Genetic CNN [28]	2017	Genetic CNN
HierarchicalEvo [29]	2018	Hierarchical Representations for Efficient Architecture Search
AE-CNN [24]	2019	Completely automated CNN architecture design based on blocks
EvoCNN [14]	2019	Evolving deep convolutional neural networks for image classification
NSGA-Net [33]	2019	NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm
RegularizedEvo [35]	2019	Regularized Evolution for Image Classfier Architecture Search
CNN-GA [32]	2020	Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification

because users can use these distributed toolkits to evaluate multiple DNNs simultaneously during the fitness evaluation phase of ENAS. Under this strategy, one GPU is used to evaluate only one DNN and there is little communication overhead incurred between GPUs. However, these existing distributed NAS toolkits, such as NNI, are complex to configure and have high learning costs. Therefore, they are not necessarily friendly for in-lab users.

In this article, we aim to develop a benchmarking platform of ENAS algorithms named BenchENAS, to address all the issues aforementioned. In summary, the contributions of the proposed BenchENAS are shown as follows.

1) We propose a platform for fair comparisons between different ENAS algorithms. Nine representative state-of-the-art ENAS algorithms, popular data processing techniques for three widely used benchmark datasets, as well as configurable trainer settings, such as learning rate, optimizers, batch size, and training epochs, have been well implemented into the proposed BenchENAS platform. To this end, the researchers can show the superiority of their proposed algorithms by making fair comparisons with the state-of-the-art ENAS algorithms. Furthermore, these algorithms cover fixed-length encoding strategies and variable-length encoding strategies, and also single-objective optimization algorithms and multiobjective optimization algorithms as shown in Table I. We believe the proposed BenchENAS platform can be a valuable tool for ENAS researchers.
2) We analyze the advantages and disadvantages of the ENAS algorithms implemented on BenchENAS by performing fair comparison experiments with popular settings. The experimental findings results show that the comparison results within BenchENAS are appreciably different from those reported in the respective original papers, implying that unfair comparisons do exist in the recent published works. This, in turn, justifies the necessity of the proposed BenchENAS platform. These experimental results can also be used by researchers as benchmark data for future studies so that they will not need to rerun these algorithms for their comparisons.
3) We design a novel and generic efficient evaluation method tailored for the population characteristics of EC. Specifically, the efficient parallel strategy and the population memory strategy are developed to accelerate the fitness evaluation phase. The parallel strategy is specifically designed for ENAS algorithms for conveniently performing parallel training of DNNs by in-lab users, which is based on the parallel training mechanism of existing deep learning libraries and can be jointly used to collectively speed up the running of the corresponding ENAS algorithms. The parallel strategy mimics the use of manual GPU assignments in the lab environment to automatically detect the available GPU in the environment and flexibly assign the free GPUs to the algorithms regardless of the GPU type. The population memory strategy, on the other hand, is used to record the fitness values for each architecture and to reuse the fitness values in the cache when an individual of the same architecture appears. This strategy avoids repeated evaluation of the same individuals, thus further reducing the waste of computational resources.
4) BenchENAS has easy extensibility and good usability. BenchENAS is easy to extend due to its highly modular design. Users can easily implement their own ENAS algorithms within BenchENAS. It is easy for users to extend dataset settings as they become available, such as benchmark datasets, data processing techniques, trainer settings (e.g., learning rate policy and optimizers), etc. BenchENAS is easy to use for four main reasons. First, BenchENAS is implemented in python using very few third-party libraries for easy installation. Second, all the ENAS algorithms in BenchENAS can be easily configured with different data settings and different trainer settings. Third, BenchENAS designs a downtime restart strategy to reboot the platform in the event of an unexpected stop to improve the stability and robustness. Finally, BenchENAS is fully open sourced and is promised to be free for research use.

The remainder of this article is organized as follows. The related works of BenchENAS are reviewed in Section II. The details of BenchENAS are presented in Section III. Section IV illustrates the extensibility and usability of BenchENAS. Section V introduces the experimental result of BenchENAS. Finally, the conclusion and future work are given in Section VI.

II. RELATED WORKS

In this section, the related works of BenchENAS are presented. Specifically, the background of ENAS algorithms is briefly documented in Section II-A, and then existing fair comparison approaches are introduced in Section II-B, and finally, popular efficient evaluation methods are shown in Section II-C.

A. Background of ENAS

NAS aims to automatically generate well-performing DNN architectures from a predefined search space using a well-designed search strategy. Mathematically, the NAS is generally considered as an optimization problem, which is formulated by

$$\begin{cases} \arg\max_A = \mathcal{P}(A, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}) \\ \text{s.t.} \quad A \in \mathcal{A} \end{cases} \tag{1}$$

where $\mathcal{A}$ denotes the search space of the neural architectures, and $\mathcal{P}(\cdot)$ measures the performance of the architecture A on the validation dataset $\mathcal{D}_{\text{valid}}$ after being trained on the training dataset $\mathcal{D}_{\text{train}}$. Essentially, NAS is a complex optimization problem experiencing multiple challenges, such as complex constraints, discrete representations, bilevel structures, computationally expensive characteristics, and multiple conflicting objectives [24].

As introduced in Section I, ENAS is a subcategory of NAS. ENAS solves the optimization problem by EC, such as GA, PSO, and DE. Specifically, GA iteratively generates new populations and eliminates poorly performing individuals based on fitness. PSO optimizes a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search space according to simple mathematical formula over the particle's position and velocity. DE operates through a similar flowchart as employed by GA. However, DE employs difference of the individuals to explore the objective function landscape. The main steps of an ENAS algorithm followed by the standard flowchart of an EC method are shown as follows.

Step 1: Initialize a population of individuals representing different DNN architectures within the predefined search space.
Step 2: Evaluate the fitness of each DNN architecture.
Step 3: Select the parent solutions from the population based on the fitness values.
Step 4: Generate offspring using evolutionary operators.
Step 5: Go to step 6 when the criterion is satisfied; otherwise, go to step 2.
Step 6: Terminate the evolutionary process and output the DNN with the best fitness value.

As can be seen above, the ENAS algorithm follows the standard flowchart of an EC method [12]. Specifically, step 2 shows the fitness evaluation phase of the ENAS algorithms. For the majority of ENAS algorithms [14], [24], [32], [35], the fitness value of each individual in the population will be estimated during the phase of fitness evaluation. In ENAS, the fitness value generally represents the performance of DNNs such as the accuracy on image classification tasks. In step 3, the mating selection operator acts as a natural selection increasing the quality of individuals because the individuals with superior performance are chosen as parent solutions to generate more competitive individuals, while ill-fitted individuals are eliminated. There are a lot of strategies in how to select an individual, such as random selection, tournament selection [29], and the roulette wheel selection [49]. In step 4, the evolutionary operators mainly include crossover and mutation. These operators create necessary diversity as well as novelty. Crossover recombines the evolutionary information of two parents to generate new offspring. Mutation, on the other hand, randomly changes the encoding of the DNN with a certain probability.

Generally, the earliest work of ENAS is viewed as the LargeEvo algorithm. Since the success of LargeEvo, ENAS has gained great attention. More and more ENAS algorithms are proposed recently. Vast neural architectures automatically designed by these ENAS algorithms have surpassed manual designs in many tasks [35], [50]. Despite the positive results of the existing ENAS algorithms, there are still some challenges and issues that need to be addressed, such as the fair comparison issue and the efficient evaluation mentioned above.

B. Fair Comparison Methods

As have introduced in Section I, the fair comparison issue may mislead researchers. More and more researchers are noticing the existence of the fair comparison issue in the NAS field and proposing some solutions. Representative works mainly include NAS-Bench-101 [51] and NAS-Bench-201 [52].

Specifically, these works enable fair comparisons by providing a benchmark architecture dataset for researchers. NAS-Bench-101 provides the first public architecture dataset for NAS Research. It constructed a search space, exploiting graph isomorphisms to identify 423K unique convolutional architectures. Then, they trained and evaluated all of these architectures three times on CIFAR-10 and compiled the results into a large dataset of over 5 million trained architectures. Each architecture can query the corresponding metrics, including test accuracy, training time, etc., directly in the dataset without the large-scale computation. NAS-Bench-201 is proposed recently and is based on cell-based encoding space. Compared with NAS-Bench-101, which was only tested on CIFAR-10, this dataset collects the test accuracy on three different image classification datasets (i.e., CIFAR-10, CIFAR-100 [44], and ImageNet-16-120 [53]). However, the encoding space is relatively small and only contains 15.6K architectures. Nevertheless, experiments with different ENAS methods on these benchmark datasets can obtain fair comparisons between different ENAS algorithms and it will not take too much time.

Unfortunately, those works have many limitations in practical applications. First, NAS-Bench-101 and NAS-Bench-201 are applicable to only a few datasets. Second, the search space of these works is fixed with a limited number of nodes and edge types in a cell. These works are only based on the cell-based encoding space and do not apply to methods based on other coding spaces.

C. Efficient Evaluation Methods

As have introduced above, the ENAS algorithms usually consume a lot of computational resources and time. In the context of labs that typically have multiple GPUs, in-lab users usually evaluate DNNs by two distributed training methods. Specifically, these two methods are using in-house

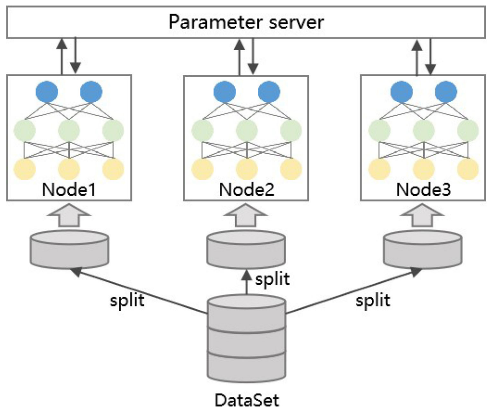


Fig. 1. Architecture of the data parallelism.

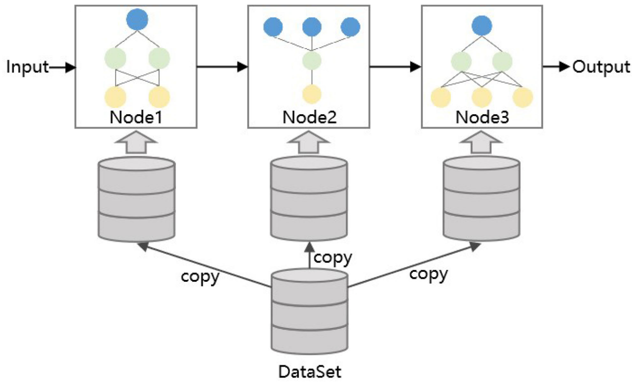


Fig. 2. Architecture of the model parallelism.

distributed parallel methods in deep learning libraries and using distributed NAS toolkits.

The in-house distributed parallel methods in deep learning libraries can be divided into two different categories: 1) data parallelism and 2) model parallelism. In the data parallelism, the dataset is split into multiple subdatasets. Every node hosts a copy of the DNN and trains the DNN on a subset of the dataset as shown in Fig. 1. Then, the values of the parameters are sent to the parameter server. After collecting all the parameters, they are averaged. This classical realization of the data parallelism is named synchronous stochastic gradient descent (SSGD) [54], [55]. This method is not efficient because the nodes are forced to wait for the slowest one at each iteration. Another method is called asynchronous SGD (ASGD) [56], [57]. This method improves on SSGD by sending outdated parameters out of the network. However, this method leads to a gradient staleness problem, which may result in slow convergence speed or even nonconvergence of the model. In the model parallelism, each node hosts a partition of the DNN, and the dataset needs to be copied to all nodes as shown in Fig. 2. The communication happens between computational nodes when the input of a node is from the output of the other computational node. The model parallelism is not commonly used because the communication expense is much higher. In addition, these distributed parallel methods are mainly targeted at a single large-scale DNN while a large number of DNNs are needed to be evaluated simultaneously in ENAS. Specifically, assuming that there are N GPUs in a lab, these methods only support accelerating the training of a single DNN with these N GPUs. In this case, however, using data parallelism can lead to synchronization overhead or cause gradient staleness problems. Using model parallelism can lead to unnecessary communication overhead. In the context of a large number of DNNs to be trained simultaneously, it is more efficient to train N individuals simultaneously with N GPUs to avoid unnecessary overhead.

In fact, some distributed NAS toolkits exist that support users to train N DNNs simultaneously with N GPUs. Representative works mainly include NNI[1] developed by Microsoft and Katib[2] at Google. Specifically, NNI has a built-in lightweight distributed training platform and trains multiple DNNs in parallel. It also supports the configuration of OpenPAI,[3] Kubernetes [58], and some other distributed scheduling platforms for distributed training of multiple DNNs. Katib only supports the configuration of Kubernetes. However, those toolkits are not friendly to in-lab users. First, they are not easy to extend. Commonly, the GPUs in the lab are distributed across multiple machines. To use the GPUs on multiple machines, those toolkits need to be installed on every machine. Second, these toolkits are more difficult to learn. In order to use some distributed functions of those toolkits, users may need to learn to use existing distributed platforms (Kubernetes and OpenPAI), such as deploying the platform, configuring NVIDIA plug-ins, setting up storage servers, and many other operations.

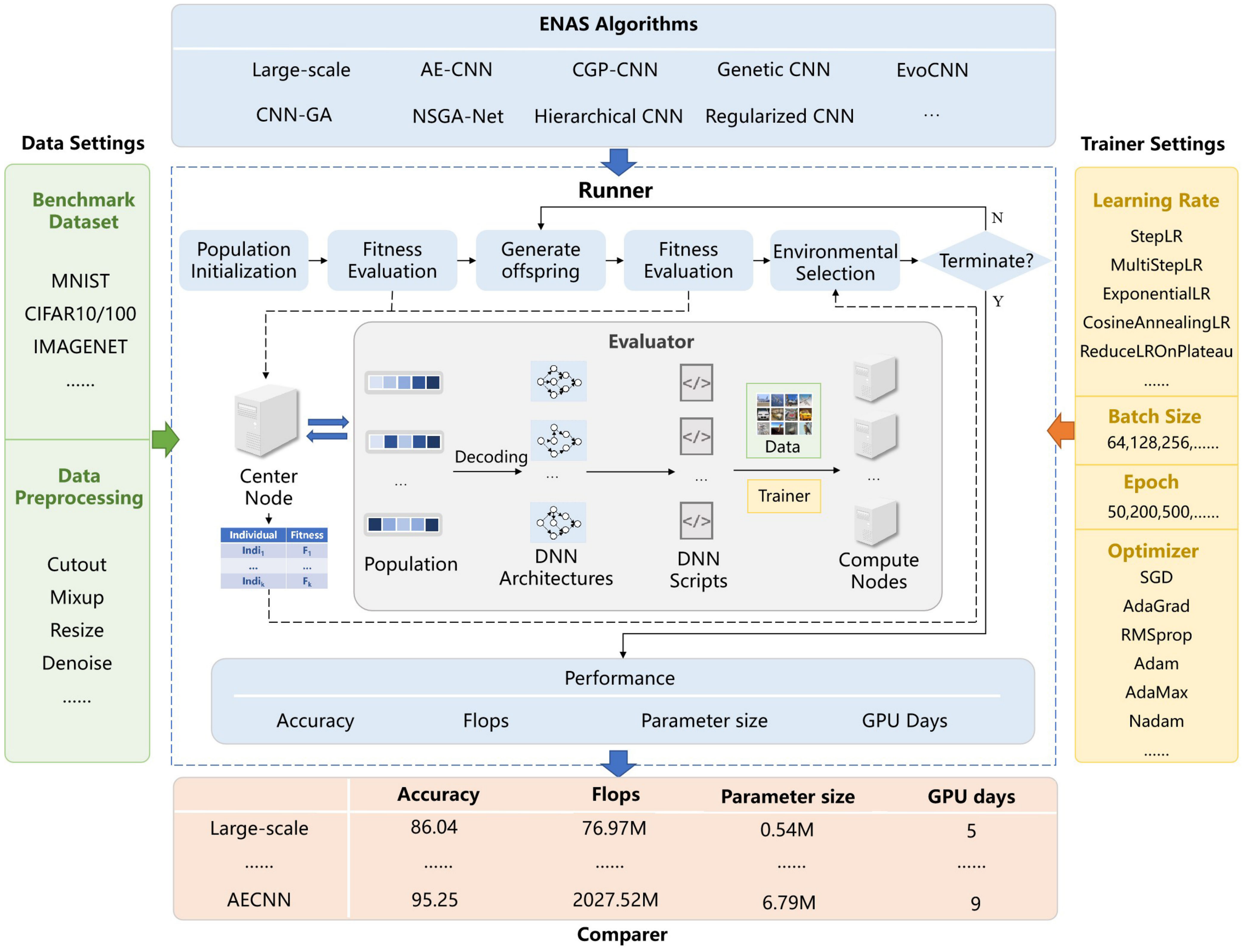
In summary, the available fair comparison methods and the efficient evaluation methods bear some serious shortcomings. The current fair comparison methods are very limited and do not apply to all the main types of ENAS. In the current efficient evaluation methods, the built-in distributed parallel methods do not take into account the population characteristics of ENAS and instead increases the overhead. The distributed NAS toolkits are complicated to configure and use, and are not suitable for in-lab users.

III. Proposed BenchENAS

A. Overview

Note that any EC-based NAS can be used within BenchENAS, and Fig. 3 shows the overview by taking the GA-based NAS as an example. BenchENAS is composed of five parts. They are *ENAS_algorithms*, *data_settings*, *trainer_settings*, *runner*, and *comparer*. After the user has selected the ENAS algorithm to run in the *ENASalgorithms* part, the data and the trainer for the training of DNNs are configured through the *data_settings* part and the *trainer_settings* part. The selected ENAS algorithm is then run through the runner part and the results are output to the *comparer* part. By running multiple ENAS algorithms with the same data settings and the same trainer settings in the *runner* part, users can get relatively fair comparison results.

[1]https://github.com/microsoft/nni
[2]https://github.com/kubeflow/katib/trial
[3]https://github.com/microsoft/pai



	Accuracy	Flops	Parameter size	GPU days
Large-scale	86.04	76.97M	0.54M	5
……	……	……	……	……
AECNN	95.25	2027.52M	6.79M	9

Fig. 3. Overview of BenchENAS.

Specifically, the *ENAS_algorithms* part includes the ENAS algorithms implemented by BenchENAS. Users can choose the algorithm they want to run or implement their own code. The *data_settings* part includes settings for the benchmark dataset and data preprocessing. Users can choose MNIST [59], CIFAR-10, CIFAR-100, or ImageNet as benchmark dataset for comparison. All of these datasets are widely used in the field of image classification. Users can also choose data preprocessing methods, such as cutout [40], mixup [39], resize, and denoise. An option is also made available to incorporate any new utilities for data preprocessing. These data preprocessing methods can be used to improve the performance of DNNs. Specifically, cutout and mixup are both commonly used data augmentation methods to improve the generalization of neural network architectures. Cutout masks out random sections of input images during training. Mixup trains a neural network on convex combinations of pairs of examples and their labels, regularizing the neural network to favor simple linear behavior in-between training examples. For the resize method, rescaling and cropping can be used to resize data. Denoising removes useless information from the data to improve the performance of neural architectures. The *trainer_settings* part includes learning rate settings, batch size settings, epoch settings, and optimizer settings. Users can set the learning rate by StepLR, MultiStepLR, ExponentialLR, CosineAnnealingLR [60], and ReduceLROnPlateau learning rate strategies. These strategies are all very practical learning rate adjustment strategies. StepLR is a strategy to adjust the learning rate at equal intervals. MultiStepLR is a strategy to adjust the learning rate at set intervals. ExponentialLR is a strategy to adjust the learning rate by an exponential decay. CosineAnnealingLR is a strategy that takes the cosine function as the period and resets the learning rate at each period. ReduceLROnPlateau is a strategy that adjusts the learning rate when a metric is no longer changing. The runner part is used to run the ENAS algorithm. As shown in Fig. 3, the center node launches and runs the entire ENAS algorithm. When the algorithm proceeds to the fitness evaluation phase, the runner uses a well-designed evaluator to obtain the fitness of each individual in the population. In the evaluator, the center node first decodes the individuals in the population into DNN architectures. Second, the center node generates DNN scripts from the DNN architectures. Third, the center node distributes the DNN scripts to the compute nodes. Each GPU in these compute nodes

Algorithm 1: Runner of BenchENAS

```
Input: Parameters required for the algorithm, the population
       size, the maximal generation number, the image dataset
       for classification, is_running.
Output: The discovered best DNN architecture.
1  if $is\_running == 0$ then
2      $t \leftarrow 0$;
3      $P_t \leftarrow$ Initialize a population;
4      $is\_running \leftarrow 1$;
5  else
6      $t \leftarrow$ Get the newest generation number;
7      $P_t \leftarrow$ Load the population;
8  end
9  Evaluate the fitness of each individual in $P_t$ using the
   evaluator;
10 Save the population $P_t$;
11 while $t <$ the maximal generation number do
12     $Q_t \leftarrow$ Generate offspring;
13     Evaluate the fitness of each individual in $P_t$ using the
       evaluator;
14     $P_{t+1} \leftarrow$Environmental selection from $P_t \cup Q_t$;
15     Save the population $P_{t+1}$;
16     $t \leftarrow t + 1$;
17 end
18 $is\_running \leftarrow 0$;
19 Return $P_t$.
```

runs a DNN script with the data and the trainer to obtain the fitness of the DNN. Finally, each compute node sends the fitness value to the center node. The comparer part includes the performance indicators, such as the accuracy, flops, parameter size, and GPU days of different ENAS algorithms. Users can do fair comparisons between different ENAS algorithms by those performance indicators. The core of BenchENAS is the runner part. The details of the implementation of the runner are given in Section III-B.

B. Runner

Runner is used to run the ENAS algorithm to get the results. The ENAS algorithm maintains a high degree of uniformity, and the implementation process is basically the same except for some minor differences in detail. To increase the stability and robustness of BenchENAS, we design a log-based downtime restart strategy to allow BenchENAS to use logs to restart when it is unexpectedly shut down. Next, we will describe in detail the operation process of the runner with the downtime restart strategy.

The pseudocode of the runner with the downtime restart strategy is shown in Algorithm 1. Specifically, for the downtime restart strategy, we add a parameter called *is_running* to the runner. "*is_running* == 0" means that the runner has not run or finished running last time. On the other hand, "*is_running* == 1" implies that the operation of the last runner was interrupted. When the runner starts running, if *is_running* is equal to 0, the population P_0 is initialized and *is_running* is set to 1 (lines 1–4). Otherwise, the latest population counter t is gotten, and the population P_t is loaded according to the acquired population number (lines 5–8). Then, the fitness of each individual, which encodes a particular architecture of the DNN, is evaluated on the given dataset by the evaluator (line 9). The population is saved (line 10). During evolution, the parent individuals are selected based on the fitness, and then a new offspring is generated by the evolutionary operators, including the crossover and mutation operators (line 12). After that, the fitness of each individual is evaluated using the evaluator (line 13). The evaluator is designed by BenchENAS to save evaluation time and will be explained in detail in the next section. Then, a population of individuals surviving into the next generation is selected by the environmental selection from the current population (line 14). Next, the population is saved (line 15). Finally, the counter is increased by one, and the evolution continues until the counter exceeds the predefined maximal generation. When the evolution is finished, the parameter *is_running* is set to 0 (line 18).

Next, we will describe how to load the population information (line 7). From Algorithm 1, we can find that the population information is saved after the fitness evaluation phase (lines 10 and 15). Specifically, BenchENAS saves the population P_t as a file named *begin_t.txt*. The file *begin_t.txt* contains the name, the encoding information, the identifier, and the fitness value of each individual in the population. When *is_running* is not 0, the latest written population file will be loaded and the information of each individual in the population will be obtained to rebuild the population P_t. The downtime restart strategy enables log-based downtime restart. Doing so gives BenchENAS the advantages of persistence and stability. When a power outage or downtime is caused by the wrong operations, the saved log files are used to restore the platform to the working state before the downtime, thus avoiding wasting computational resources by starting training from scratch.

In the next section, we will detail the implementation of the evaluator, which is well designed to save the computational resources and evaluation time.

C. Evaluator

As mentioned above, because training DNNs is very time consuming, varying from several hours to even several months depending on the particular architecture, the evaluator is designed to speed up the fitness evaluation phase in BenchENAS. In the evaluator, we reduce the evaluation time and the computational resources by the population memory strategy and the parallel strategy. Specifically, the population memory strategy is used to store the fitness of every DNN evaluated. It works by reusing the fitness of the DNNs that have previously appeared in the population to save time. The parallel strategy works by evaluating multiple individuals on multiple GPUs. Next, we will describe in detail how the evaluator uses these two strategies.

The detail of the evaluator is shown in Algorithm 2, which includes the specifics of the population memory strategy (lines 4–15). Briefly, given the population P_t containing all the individuals for evaluating the fitness, the evaluator evaluates each individual of P_t in the same manner, and finally, returns P_t containing the individuals whose fitness have been evaluated. Specifically, for each individual in the population, the evaluator first decodes the individual into DNN and generates the python script of the DNN (lines 2 and 3). The population memory strategy is achieved in lines 4–15. Specifically, if the cache does not exist, the evaluator will create an empty global cache system (denoted as *Cache*), storing the fitness of the individuals with unseen architectures (lines 4–7). Then, if the individual is found in *Cache*, its fitness is directly derived from *Cache* (lines 8–10). Otherwise, the individual is asynchronously evaluated using the parallel strategy to obtain the fitness of the individual (lines 12 and 13). The identifier and fitness value of the individual will be stored to *Cache* (line 14). For the population memory strategy, querying an individual from *Cache* is based on the individual's identifier. Theoretically, arbitrary identifiers can be used as long as they can distinguish individuals encoding different architectures. In BenchENAS, the 224-hash code [61], which has been implemented by most programming languages in terms of the encoded architecture, is used as the corresponding identifier. For the parallel strategy, the individual is asynchronously placed on an available GPU, which implies that multiple individuals can be evaluated simultaneously on multiple GPUs.

Algorithm 2: Evaluator

```
Input: The population $P_t$ of the individuals to be evaluated.
Output: The population $P_t$ of the individuals with their fitness
        values.
1  foreach individual in $P_t$ do
2      Decode the individual to DNN;
3      Generate the python script for the DNN;
4      if Cache does not exist then
5          Cache $\leftarrow \emptyset$;
6          Set Cache to a global variable;
7      end
8      if the identifier of individual in Cache then
9          $v \leftarrow$ Query the fitness by identifier from Cache;
10         Set $v$ to individual;
11     else
12         $v \leftarrow$ evaluate individual by the parallel strategy;
13         Set $v$ to individual;
14         Cache $\leftarrow$ the identifier of individual and the fitness;
15     end
16 end
17 Return $P_t$.
```

The population memory strategy is designed to speed up the fitness evaluation phase of the ENAS algorithms, which is mainly based on the following two considerations. First, individuals who survive to the next generation without changes in the neural network architecture do not need to be reevaluated. Second, the evolutionary operators, such as crossover and mutation, may generate individuals that have been evaluated before. In such a context, the population memory strategy can be used to save time and improve evolutionary efficiency. In general, we should be concerned about the size of the population memory strategy and discuss the conflicting collision problem resulted from the duplicate keys. However, in BenchENAS, it is not necessary a concerning issue. First, the population memory strategy is similar to a map data structure. Each of these records in the population memory strategy is a string containing the identifier of a DNN and the corresponding fitness value. For example, a record such as "id = 90.50" denotes that the identifier of the DNN is "id" and its fitness value is "90.50." Second, as we mentioned in the last paragraph, the identifier is calculated by the 224-hash code that can generate 2^{224} different identifiers. Commonly, the ENAS algorithms only evaluate thousands of individuals. Obviously, the issue will hardly happen. Third, the 224-hash code implementation used by BenchENAS will generate the identifier with the length of 32, the symbol "=" with the length of 1, and the fitness value with the length of 4. Thus, each record in the population memory strategy is a string with the length of 37, occupying 37 bytes with the UTF-8 file encoding. Obviously, the population memory strategy will occupy less than 1 MB of disk space, even though there are tens of thousands of records. Therefore, we do not need to consider the size of the population memory strategy.

The parallel strategy is a parallel computing platform based on GPUs. As shown in Algorithm 2, when the identifier of the individual is not in *Cache*, the evaluator will evaluate the individual by the parallel strategy. This strategy is designed for the ENAS algorithm because the EC algorithms are population based. In this strategy, a GPU is used to evaluate only one DNN. Assuming a total of N GPUs for all compute nodes, N individuals are evaluated simultaneously during the fitness evaluation phase of ENAS. Specifically, the center node gets GPU usage from a SQL database. When the ENAS algorithm begins to run, the SQL database, including the state (i.e., usage) of each GPU, is created. The center node remotely queries the GPU usage of all compute nodes in parallel at a regular intervals. Note that the center node uses the commands provided by NVIDIA GPU driver to get the GPU usage, which is transparent to the GPU types. When the fitness of an individual is to be obtained, the evaluator will use the parallel strategy to evaluate the individual as Algorithm 3 shows. Specifically, while there is an available GPU by querying the SQL database, the center node first gets the compute node, says node$_j$, and the identifier of GPU$_k$. Second, the center node sets the usage of GPU$_k$ to busy in the SQL database. Then, the center node sends the DNN script to node$_j$ and remotely commands node$_j$ to train the DNN script with GPU$_k$ (lines 2–7). Finally, when the training of the DNN script is completed, the fitness value of the DNN script will be obtained. The status of the GPU$_k$ is updated again in the SQL database.

Next, the reasons for designing such a parallel strategy are given. Since training DNNs can take a lot of time, in-lab users often run the ENAS algorithm with multiple GPUs to speed up the evaluation. As discussed in related works, current methods of running ENAS with multiple GPUs, such as the in-house distributed parallel methods and the distributed NAS toolkits, are not suitable for in-lab users. This motivates us to design such a parallel strategy. The parallel strategy is designed using parallel computing techniques. In parallel computing, large problems can often be divided into multiple independent subproblems, which can then be solved at the same time. By parallel performing these subproblems in different computational platforms, the total processing time of the entire problem is consequently shortened. In the fitness evaluation phase of ENAS, multiple individuals are waiting to be evaluated at the same time due to the population-based

Algorithm 3: Parallel Strategy

```
  Input: The individual $indi_i$ to be evaluated.
  Output: The fitness values of $indi_i$.
1 while there is an available GPU in the SQL database do
2     $node_j \leftarrow$ get the compute node where the GPU located;
3     $GPU_k \leftarrow$ get the GPU id;
4     Set $GPU_k$ to busy in the SQL database;
5     Send the DNN script of $indi_i$ to $node_j$;
6     $Fitness_i \leftarrow$ Remote command $node_j$ train the DNN script
      with $GPU_k$;
7     Set $GPU_k$ to the idle state in the SQL database;
8 end
9 Return $Fitness_i$.
```

characteristics. Furthermore, the fitness evaluation of each individual is independent, which just satisfies the scene of using the parallel computing techniques. As a result, we design the parallel strategy. For the parallel strategy, the individual is placed in parallel on an available GPU, which implies that we do not need to wait for the fitness evaluation of the next individual until the fitness evaluation of the current one finishes, but place the next individual on an available GPU immediately. In doing so, BenchENAS significantly reduces the time consumed by the fitness evaluation.

D. Comparer

After performing comparison experiments between different ENAS algorithms with the same data settings, trainer settings, and the number of function evaluations, we can compare the experimental results of these algorithms to analyze their features and performance. Specifically, we can obtain the accuracy, the parameter size, and the GPU days of the optimal DNNs searched by the algorithms. The accuracy measures the performance of the algorithms. The parameter size measures the magnitude of the network. The GPU days measure the time complexity of the algorithms. But the comparisons are not simply made by comparing the values. This is because researchers usually make tradeoffs between accuracy, network size, and GPU days. For example, the multiobjective ENAS algorithms, such as NSGA-Net, make a tradeoff between the accuracy performance and the size of the network, and discard the performance to obtain a smaller network size, thus allowing the DNNs to run on some mobile device. One-shot learning-based methods and predictor-based methods (i.e., surrogate ENAS) both make a tradeoff between the time complexity and the performance of ENAS algorithms to greatly improve the efficiency of evaluation.

In general, we design BenchENAS to provide a platform for fair comparisons. The researchers in the field of ENAS can obtain more objective and fair evaluations of various ENAS algorithms and highlight the advantages of their proposed algorithms. Therefore, this is beneficial to the development of the ENAS field.

IV. Experiments and Analysis

In this section, we run eight ENAS methods on BenchENAS, which can serve as baselines for future research. Since the original paper of EvoCNN did not perform experiments on the CIFAR-10 dataset, it does not participate in the comparison of ENAS algorithms. Specifically, we evaluate some typical ENAS algorithms, including LargeEvo [13], Genetic CNN [28], HierarchicalEvo [29], CGP-CNN [30], AE-CNN [24], NSGA-Net [33], RegularizedEvo [35], and CNN-GA [32]. Note that NSGA-Net is a multiobjective NAS algorithm that will generate multiple Pareto-optimal solutions. In Table III, we only pick the architecture with the highest accuracy for comparison because this experiment is designed to show the best accuracy. The experiments of BenchENAS are performed on GPU cards with the same model of Nvidia GeForce GTX 2080 Ti.

To ensure a consistent evolution across ENAS algorithms, for each algorithm, the number of function evaluations is set at 400. We ensure that the other settings of the algorithm are consistent with those cited in their original papers, such as mutation probability, network length, convolution layer settings, etc. Due to the limitation of computing resources, we used the early stop policy in our experiments. We divide the algorithm into the evolution phase and the retrain phase. The evolutionary phase is the search process of ENAS, in which only the best performing DNN architecture needs to be found, so it is not necessary to train each DNN to its best performance. The approximate performance of each DNN can be observed by training epochs using the early stop policy. In the evolution phase, we use 50 epochs to train the DNNs to reduce the evolution time. In the retrain phase, we retrain the best DNN selected by the algorithm to reach its best performance, and this phase uses 600 epochs. The specific settings will be explained in Section V-B.

A. Data Settings

The CIFAR-10 benchmark dataset is chosen as the image classification task in the experiments. There are two reasons to choose it. First, the dataset is challenging in terms of the image sizes, categories of classification, and noise as well as rotations in each image. Second, it is widely used to measure the performance of ENAS algorithms, and most of the ENAS algorithms have publicly reported their classification accuracy on it. Specifically, the CIFAR-10 dataset is an image classification benchmark for recognizing ten classes of natural objects, such as airplanes and birds. It consists of 60 000 RGB images in the dimension of 32×32. In addition, there are 50 000 images and 10 000 images in the training set and the testing set, respectively. Each category has an equal number of images. In order to do a fair comparison, we employ the data preprocessing method that is often used in ENAS algorithms [2], [3], [32]. Specifically, each direction of one image is padded by four zeros pixels, and then an image with the dimension of 32×32 is randomly cropped. Finally, a horizontal flip is randomly performed on the cropped image with a probability of 0.5. In this retraining phase, we use the additionally data preprocessing technique cutout.

TABLE II
EXPERIMENTAL RESULTS OF EACH ALGORITHM IN THE *Original* PAPER ON *CIFAR-10*

Algorithm	Accuracy	#Parameters	GPU days	Number of function evaluations
LargeEvo [13]	94.60	5.4M	2750	-
CGP-CNN [30]	94.02	1.68M	27	1,000
Genetic CNN [28]	92.90	-	17	1,000
HierarchicalEvo [29]	96.37	-	300	1,400,000
AE-CNN [24]	95.7	2.0M	27	400
NSGA-Net [33]	96.15	3.3M	8	1,200
RegularizedEvo [35]	96.60	2.6M	-	20,000
CNN-GA [32]	95.22	2.9M	35	400

TABLE III
EXPERIMENTAL RESULTS OF EACH ALGORITHM ON *BenchENAS*

Algorithm	Accuracy after 50 epoch	Re-train accuracy	#Parameters	GPU days
LargeEvo	80.55	86.04	0.54M	5
CGP-CNN	89.07	94.24	4.62M	14
Genetic CNN	85.36	92.86	0.28M	3
HierarchicalEvo	85.59	95.26	47.13M	4
AE-CNN	89.04	95.25	6.79M	9
NSGA-Net	85.12	93.08	1.08M	4
RegularizedEvo	89.97	94.42	7.88M	15
CNN-GA	90.85	94.67	2.86M	5

B. Trainer Settings

To assign the fitness to the candidate DNN architectures, we train the DNN by stochastic gradient descent (SGD) with a momentum of 0.9, a minibatch size of 64, and a weight decay of 5.0×10^{-4}. The softmax cross-entropy loss is used as the loss function. We train each DNN for 50 epochs at an initial learning rate of 0.025.

After the evolution process, we retrain the best DNN architecture. In this retraining phase, we optimize the weights of the best architecture for 600 epochs with a different training procedure. We use SGD with a momentum of 0.9, a minibatch size of 96, and a weight decay of 3.0×10^{-4}. We start with a learning rate of 0.025 and use the cosineAnnealing learning rate.

C. Result and Analysis

In our experiments, we mainly compare the classification accuracy, parameter size, and GPU days for different ENAS algorithms. For the convenience of summarizing the comparison results, we use the name of the ENAS algorithms as the name of the discovered best DNN when comparing the classification accuracy, parameter size, and GPU days between different ENAS algorithms.

The accuracy, parameter size, GPU days, and the number of function evaluations in the original paper are shown in Table II. The accuracy after 50 epochs on the test dataset, retrain accuracy, parameter size, and GPU days of these ENAS algorithms in BenchENAS are shown in Table III. The number of function evaluations of these algorithms is 400. The symbol "−" implies there is no result publicly reported by the corresponding algorithm.

We can find from the experimental results that in terms of accuracy, the accuracy ranking of all algorithms derived from the original papers is: RegularizedEvo >HierarchicalEvo >NSGA-Net >AECNN >CNN-GA >LargeEvo >CGP-CNN >Genetic CNN with RegularizedEvo ranked number one while Genetic CCN ranked the last. On the other hand, the accuracy ranking of all algorithms under the same experimental conditions in the proposed BenchENAS platform is: HierarchicalEvo >AECNN >CNN-GA >RegularizedEvo >CGP-CNN >NSGA-Net >Genetic-CNN >LargeEvo. It can be seen that the performance of each ENAS algorithm under the same experimental conditions is not the same as quoted from the original papers. The comparability problem does exist in the comparison of the algorithms. Next, we will analyze why the situation that leads to different performance from the original paper occurs.

From the two rankings, we can observe that RegularizedEvo, NSGA-Net, and LargeEvo rankings are different from the ones in the original paper. The LargeEvo algorithm evolved from a linear structure in the original paper, starting with 1000 individuals per generation and evolving for 11 days to obtain a classification accuracy of 94.60%. In this experiment, the LargeEvo algorithm stopped evolving after searching only 400 individuals. Therefore, the obtained neural network is a shallow one with poor performance. For NSGA-Net, we believe that it is the result of the combination of different data settings, different trainer settings, and different number of function evaluations. In the original paper of NSGA-Net, it incorporates a data preprocessing technique cutout [40], and a regularization technique, scheduled path [62]. To further improve the training process, an auxiliary head classifier is also appended to the DNN search. To ensure the same setup for each experiment, these tricks are not applied in our experiments. In addition, the number of function evaluations of NSGA-Net is 1000 in the original paper, while the number is only 400 in this experiment. These factors together cause NSGA-Net to perform less satisfactorily than the original paper showed. For the RegularizedEvo algorithm, the number of function evaluations of the original paper is 20 000 while the number is only 400 in this experiment. In addition, the RegularizedEvo algorithm uses the model augmentation trick in [62], and RMSProp optimizers and the scheduled path in training that are not used in this experiment. As a result, the performance is not as good as the original paper indicated. In conclusion, comparing the results of the original paper with the results on BenchENAS, we can see that unfair comparisons do exist. First, some tricks can improve the accuracy of the best DNNs obtained by NAS, which is very unfair for the algorithms that do not use tricks. Second, in ENAS, some algorithms evaluate a large number of function evaluations, which is not fair for algorithms that evaluate a small number of function evaluations. Finally, in the original paper, different algorithms use different GPU models, and the algorithm that uses GPUs with better performance will get better results, which will also cause unfair comparisons.

Given the obtained results from BenchENAS, we can analyze the features of various ENAS algorithms more fairly and more objectively. We analyze these algorithms in descending order of accuracy. Specifically, LargeEvo starts from a linear architecture and does an evolutionary process from the primitive convolution operation. Compared with other ENAS algorithms that start from deep network architectures, LargeEvo obtains the worst accuracy under the same settings. This evolutionary process from scratch is unaffordable for most researchers. For Genetic CNN, it obtains better accuracy compared with LargEvo due to the more refined encoding strategy compared with LargeEvo. However, it uses a fixed-length encoding strategy, and searches for CNNs with shallow depths. The shallow depths of CNN lead to the minimum parameter size, minimum GPU days, and low accuracy. NSGA-Net is a multiobjective algorithm to make a tradeoff between accuracy and network size, thus achieving a smaller parameter size and a better accuracy. CGP-CNN, RegularizedEvo, and CNN-GA obtain similar accuracy. These algorithms search DNNs in a designed search space. Among them, CNN-GA demonstrates its superiority through the highest accuracy, smallest parameter size, and smallest GPU days. Although RegularizedEvo has higher accuracy compared with CGP-CNN, it also has a larger parameter size and GPU days. HierachicalEvo and AE-CNN obtain similar accuracy. However, the parameter size of AE-CNN is much smaller. This is because the size of architectures in the search space of HierarchicalEvo is larger.

Therefore, we believe that BenchENAS is effective for fair comparisons of ENAS algorithms. Through the training results of this platform, users can analyze the advantages and disadvantages of each algorithm more objectively without being influenced by other conditions, such as trainer settings, data settings, and the number of function evaluations. Besides, we can see that the experiments on BenchENAS reduce the consumption of GPU resources, yet still make a valid and reasonable comparison.

V. Conclusion and Future Work

This article has introduced a benchmarking platform for ENAS, named BenchENAS. Version 1.0 of BenchENAS includes nine EC-based NAS algorithms. BenchENAS is completely open sourced, such that interested users are able to develop new algorithms on top of it. In a nutshell, BenchENAS provides a platform for the fair comparisons of different ENAS algorithms. Furthermore, BenchENAS elaborates an evaluator to speed up the fitness evaluation and save much computational resource by the population memory strategy and the parallel strategy. BenchENAS has easy extensibility and good usability so that users can easily extend their own algorithms and easily use the platform. We have done comparative experiments on BenchENAS using eight state-of-the-art ENAS algorithms to demonstrate that the fair comparison issue is possible and to provide benchmark data for future studies by researchers.

However, there are still issues in BenchENAS to be attended to. First, the number of implemented ENAS algorithms remains small. Second, the efficient evaluator designed in BenchENAS only alleviates the problem of expensive computational resources, but does not solve the problem completely. Finally, the search spaces of different ENAS algorithms are still different at present. As expected, there is still room for improvement for a truly fair comparison.

In the future, we will keep trying to solve the above problems. First, although BenchENAS allows the users to submit their own codes to be included, we will keep following and adding more state-of-the-art ENAS algorithms into BenchENAS. The authors with promising ENAS algorithms will also be actively solicited. Second, some efficient methods for evaluating DNNs already exist to solve the problem of time-consuming DNN evaluation, and we will consider incorporating them into BenchENAS. Finally, we will exploit means to make all algorithms compare in the same search space. Furthermore, we will continuously maintain and develop BenchENAS for years to come.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 4700–4708.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” 2018, *arXiv:1810.04805*.

[5] Y. Zhang, W. Chan, and N. Jaitly, “Very deep convolutional networks for end-to-end speech recognition,” in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, New Orleans, LA, USA, 2017, pp. 4845–4849.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[7] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2017, pp. 5998–6008.

[8] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.

[9] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996.

[11] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.

[12] T. Bäck, D. B. Fogel, and Z. Michalewicz, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997.

[13] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[14] 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, Apr. 2020.

[15] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[16] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. San Francisco, CA, USA: Morgan Kaufmann Publ. Inc., 1998.

[17] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[18] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 4. Perth, WA, Australia, 1995, pp. 1942–1948.

[19] Y. Sun, G. G. Yen, and Z. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 173–187, Apr. 2019.

[20] 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, Apr. 2002.

[21] Y. Sun, G. G. Yen, and Z. Yi, "Improved regularity model-based EDA for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 662–678, Oct. 2018.

[22] M. Jiang, Z. Huang, L. Qiu, W. Huang, and G. G. Yen, "Transfer learning-based dynamic multiobjective optimization algorithms," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 501–514, Aug. 2018.

[23] Y. Sun, G. G. Yen, and Z. Yi, "Reference line-based estimation of distribution algorithm for many-objective optimization," *Knowl. Based Syst.*, vol. 132, pp. 129–143, Sep. 2017.

[24] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

[25] 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.

[26] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl. Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[27] H. Al-Sahaf *et al.*, "A survey on evolutionary machine learning," *J. Roy. Soc. New Zealand*, vol. 49, no. 2, pp. 205–228, 2019.

[28] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Venice, Italy, 2017, pp. 1379–1388.

[29] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017, *arXiv:1711.00436*.

[30] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 497–504.

[31] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *Proc. 10th Annu. Conf. Companion Genet. Evol. Comput.*, 2008, pp. 2701–2726.

[32] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[33] Z. Lu *et al.*, "NSGA-net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 419–427.

[34] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proc. Int. Conf. Parallel Problem Solving Nat.*, 2000, pp. 849–858.

[35] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[36] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 6, 2021, doi: 10.1109/TNNLS.2021.3100554.

[37] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. 35th Int. Conf. Uncertainty Artif. Intell.*, 2020, pp. 367–377.

[38] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," 2019, *arXiv:1902.08142*.

[39] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2017, *arXiv:1710.09412*.

[40] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017, *arXiv:1708.04552*.

[41] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "DropBlock: A regularization method for convolutional networks," 2018, *arXiv:1810.12890*.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 2818–2826.

[43] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training imagenet in 1 hour," 2017, *arXiv:1706.02677*.

[44] A. Krizhevsky and G. Hinton. "Learning Multiple Layers of Features From Tiny Images." 2009. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html

[45] L. M. Schmitt, "Theory of genetic algorithms," *Theor. Comput. Sci.*, vol. 259, nos. 1–2, pp. 1–61, 2001.

[46] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY, USA: Oxford Univ. Press, 1996.

[47] A. Paszke *et al.*. "Automatic differentiation in PyTorch," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2017, pp. 1–4. [Online]. Available: https://openreview.net/pdf?id=BJJsrmfCZ

[48] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2016, pp. 265–283.

[49] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," M.S. thesis, Dept. Comput. Commun. Sci., Univ. Michigan, Ann Arbor, MI, USA, 1975.

[50] V. Passricha and R. K. Aggarwal, "PSO-based optimized CNN for Hindi ASR," *Int. J. Speech Technol.*, vol. 22, no. 4, pp. 1123–1133, 2019.

[51] 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.*, 2019, pp. 7105–7114.

[52] X. Dong and Y. Yang, "NAS-bench-201: Extending the scope of reproducible neural architecture search," 2020, *arXiv:2001.00326*.

[53] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*.

[54] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of DNNs with natural gradient and parameter averaging," 2014, *arXiv:1410.7455*.

[55] S. Shi *et al.*, "A distributed synchronous SGD algorithm with global top-k sparsification for low bandwidth networks," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 2238–2247.

[56] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-SGD for distributed deep learning," 2015, *arXiv:1511.05950*.

[57] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3043–3052.

[58] D. Bernstein, "Containers and cloud: From LXC to docker to Kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.

[59] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[60] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," 2016, *arXiv:1608.03983*.

[61] R. Housley, "A 224-bit one-way hash function: Sha-224," IETF, RFC 3874, Sep. 2004.

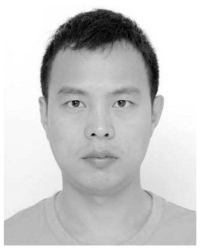
[62] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 8697–8710.

Xiangning Xie received the B.E. degree in software engineering from Sichuan University, Chengdu, China, in 2021, where she is currently pursuing the Ph.D. degree with the College of Computer Science.

Her main current research interests include neural architecture search and evolutionary algorithm.

Yuqiao Liu (Graduate Student Member, IEEE) received the B.Eng. degree in computer science and technology from Sichuan University, Chengdu, China, in 2020, where he is currently pursuing the M.S. degree with the College of Computer Science.

His current research interests include neural architecture search and evolutionary algorithm.



Yanan Sun (Member, IEEE) received the Ph.D. degree in engineering from Sichuan University, Chengdu, China, in 2017.

He is currently a Research Professor with Sichuan University, Chengdu, China. Prior to that, he has been a Postdoctoral Research Fellow with the Victoria University of Wellington, Wellington, New Zealand, from July 2015 to March 2019. He has published 338 peer-reviewed papers including 18 papers in IEEE transactions journals. As the PI, he has received two research grants from Science & Technology Department of Sichuan Province, and one from National Natural Science Foundation of China.

Dr. Sun received the Best Student Paper Award of IEEE CIS Chengdu Chapter, the National Scholarship of China, and the IEEE Student Travel Grant in 2016. He is the Leading Organizer of one workshop and one special session in the topic of Evolutionary Deep Learning, and the Founding Chair of IEEE CIS Task Force on Evolutionary Deep Learning and Applications. He is also the Guest Editor of the Special Issue on Evolutionary Computer Vision, Image Processing, and Pattern Recognition in Applied Soft Computing.

Gary G. Yen (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Notre Dame, Notre Dame, IN, USA, in 1992.

He is currently a Regents Professor with the School of Electrical and Computer Engineering, Oklahoma State University (OSU), Stillwater, OK, USA. Before joining OSU in 1997, he was with the Structure Control Division, U.S. Air Force Research Laboratory, Albuquerque, NM, USA.

Dr. Yen received the Andrew P Sage Best Transactions Paper Award from IEEE Systems, Man and Cybernetics Society in 2011 and the Meritorious Service Award from IEEE Computational Intelligence Society in 2014. He was an Associate Editor of the *IEEE Control Systems Magazine*, IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, *Automatica*, *Mechantronics*, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, and IEEE TRANSACTIONS ON NEURAL NETWORKS. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON CYBERNETICS. He served as the General Chair for the IEEE International Symposium on Intelligent Control held in Houston, TX, USA, and 2006 IEEE World Congress on Computational Intelligence held in Vancouver, Canada. He served as the Vice President for the Technical Activities from 2005 to 2006 and then the President of the IEEE Computational Intelligence Society from 2010 to 2011. He was the Founding Editor-in-Chief of the *IEEE Computational Intelligence Magazine* from 2006 to 2009.

Bing Xue (Senior Member, IEEE) received the B.Sc. degree from the Henan University of Economics and Law, Zhengzhou, China, in 2007, the M.Sc. degree in management from Shenzhen University, Shenzhen, China, in 2010, and the Ph.D. degree in computer science from the Victoria University of Wellington (VUW), Wellington, New Zealand, in 2014.

She is currently a Professor of Computer Science and the Program Director of Science with the School of Engineering and Computer Science, VUW. She has over 300 papers published in fully refereed international journals and conferences. Her research focuses mainly on evolutionary computation, machine learning, classification, symbolic regression, feature selection, evolving deep NNs, image analysis, transfer learning, and multiobjective machine learning.

Dr. Xue is currently the Chair of IEEE CIS Task Force on Transfer Learning & Transfer Optimization, the Vice-Chair of IEEE CIS Evolutionary Computation Technical Committee, IEEE Task Force on Evolutionary Feature Selection and Construction, and IEEE CIS Task Force on Evolutionary Deep Learning and Applications, and an Editor of IEEE CIS Newsletter. She has also served as an Associate Editor of several international journals, such as *IEEE Computational Intelligence Magazine*, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, and IEEE TRANSACTIONS ON ARTIFICIAL INTELLIGENCE.

Mengjie Zhang (Fellow, IEEE) received the B.E. and M.E. degrees from the Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, in 1989 and 1992, respectively, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2000.

He is currently a Professor of Computer Science, the Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) of the Faculty of Engineering, Victoria University of Wellington, Wellington, New Zealand. He has published over 700 papers in refereed international journals and conferences. His current research interests include evolutionary computation, particularly genetic programming and particle swarm optimization with application areas of image analysis, multiobjective optimization, feature selection and reduction, job shop scheduling, and transfer learning.

Prof. Zhang was the Chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, the IEEE CIS Emergent Technologies Technical Committee, and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is the Vice-Chair of the IEEE CIS Task Force on Evolutionary Feature Selection and Construction and the Task Force on Evolutionary Computer Vision and Image Processing, and the Founding Chair of the IEEE Computational Intelligence Chapter in New Zealand. He is also a Committee Member of the IEEE NZ Central Section. He is a Fellow of the Royal Society of New Zealand and an IEEE CIS Distinguished Lecturer. He has been a Panel Member of the Marsden Fund (New Zealand Government Funding).