sciendo

JAISCR, 2025, Vol. 15, No. 4, pp. 413 – 446
10.2478/jaiscr-2025-0020

AN EVOLUTIONARY NEURAL ARCHITECTURE SEARCH METHOD ACCELERATED BY MULTI-FIDELITY EVALUATION AND GENETIC DECISION CONTROLLER

Zenglin Qiao[1,2], Ali Asghar Heidari[3], Xinchao Zhao[1,2,*], Huiling Chen[4,*]

[1]*School of Mathematical Sciences,*
Beijing University of Posts and Telecommunications,
Beijing 100876, China

[2]*Key Laboratory of Mathematics and Information Networks*
(Beijing University of Posts and Telecommunications),
Ministry of Education, China

[3]*School of Surveying and Geospatial Engineering, College of Engineering,*
University of Tehran, Tehran, Iran

[4]*Department of Computer Science and Artificial Intelligence,*
Wenzhou University, Wenzhou 325035, China

[*]*E-mail: zhaoxc@bupt.edu.cn and chenhuiling.jlu@gmail.com*

Submitted: 23rd April 2025; Accepted: 20th June 2025

Abstract

Deep neural networks (DNNs) are now widely used in numerous fields. However, manually design-ing DNNs is labor-intensive and requires expert knowledge. Evolutionary neural architecture search (ENAS) is an efficient method. Nevertheless, ENAS requires full-epoch training for each candidate architecture to determine fitness values, leading to high computational costs. To accelerate the search process and reduce resource consumption, this paper proposes MFGENAS, an accelerated ENAS method. MFGENAS is implemented within the NSGA-II framework and employs two key acceleration strategies: multi-fidelity evaluation and a genetic decision controller. We demonstrate through experi-mental analysis that classification-based prediction is significantly more effective than regression-based prediction in estimating architecture performance. To reduce the need for expensive evaluations, we introduce a genetic decision controller to evaluate the quality of generated offspring. This process is treated as a classification task: if the controller predicts that the offspring will outperform the parent, the offspring is retained. For this purpose, we adopt a kernel extreme learning machine optimized by an improved polar lights optimizer as the genetic decision controller. Comprehensive experiments vali-date MFGENAS's performance. On the CIFAR-10 dataset, MFGENAS discovers an architecture with an error rate of 2.39% using only 0.3 GPU days. In contrast, the classical AE-CNN requires 27 GPU days to achieve an architecture with a higher error rate of 4.30%. On the CIFAR-100 dataset, MFGENAS discovers an architecture with an error rate of 16.42% in just 0.3 GPU days, whereas E2EPP requires 8.5 GPU days to reach an error rate of 22.02%.

Keywords: evolutionary neural architecture search; evolutionary algorithm; multi-fidelity evaluation; polar lights optimizer; convolutional neural networks

1 Introduction

Currently, deep learning technologies have achieved significant success across various fields and are widely applied to address numerous machine learning tasks, including classification [1, 2], object detection [3, 4], speech recognition [5], and natural language processing [6]. The performance of deep learning models heavily relies on the structure and parameters of deep neural networks. Different deep neural networks yield varying performance levels when handling distinct machine learning tasks. As a result, neural networks that balance high performance with low complexity are highly sought after. Researchers have developed many high-performing deep neural networks, such as ResNet [7], DenseNet [8], and GoogLeNet [9], which find extensive applications in multiple domains. However, these classic neural networks are typically designed manually by experts based on experience, often relying on specialized knowledge and requiring continuous parameter adjustments to achieve optimal performance. This process is time-consuming and challenging for ordinary researchers. Such demands have sparked interest in the automatic design of deep neural networks, leading to the emergence of a popular research area known as NAS.

The early NAS tasks were proposed by Zoph et al. [10], utilizing reinforcement learning techniques to search for suitable convolutional neural network architectures. They later introduced the classic NAS [10], which can discover competitive neural network architectures with those designed manually. However, early NAS methods employed reinforcement learning; over the years, various optimization techniques have been applied to NAS, including reinforcement learning [11], gradient optimization [12], Bayesian optimization [13, 14], and evolutionary algorithms (EA) [15, 1]. In fact, NAS tasks can be viewed as a combinatorial optimization problem for deep neural networks. Due to the enormous search space, each method demonstrates effectiveness but also has its drawbacks.

Given the complexity of NAS tasks and the vast search space, ENAS has gained significant attention [16-18]. Inspired by biological evolution, EA are optimization techniques well-suited for solving complex problems. These algorithms continuously generate and refine solutions by simulating mechanisms such as natural selection, inference, and biological mutation. Typical EA includes genetic algorithm and differential evolution, which generally involve initializing a random solution set, evaluating the fitness of solutions, selecting high-performing individuals for crossover and mutation, and forming a new solution set. A notable advantage of EA is their robust global search capability, effectively avoiding nonlinear local optima, making them particularly suitable for nonlinear and complex multi-objective optimization problems. For example, Xu et al. [19] employed a particle swarm optimization (PSO) algorithm for NAS, incorporating a scale-adaptive initialization strategy to enhance feature extraction capabilities. The evaluation metrics included model parameter size, floating-point operations (FLOPs), and classification accuracy, which were jointly optimized as a multi-objective formulation. Experimental results on cancer case images demonstrated that the proposed NAS method could discover high-quality deep neural network architectures. Wen et al. [20] employed a genetic algorithm for NAS, with distinctive designs in the search space, encoding scheme, and population update operators. Experimental results demonstrated that the proposed ENAS method achieved an accuracy of 96.35% on the CIFAR-10 dataset, while requiring only 0.2 GPU days for the search process.

Although the architectures discovered by ENAS demonstrate strong performance, the reliance of EA on continuous population updates to identify optimal architectures leads to significant computational resource and time expenditures for the complete performance evaluation of each individual in the population. Consequently, ENAS typically incurs high costs, hindering its widespread adoption in practical applications. Several effective methods have been proposed within the ENAS framework to address this issue. These methods can be broadly categorized into two types. The first category focuses on reducing the number of evaluations required for individual architectures, as complete evaluations are a primary factor contributing to the time consumption of ENAS. Key techniques in this category include surrogate models [21], early stopping [22], and population reduction with memory [15]. The second category aims to decrease the computational costs associated with evaluating individual architectures, employing strate-

gies such as weight inheritance [23], subset training [24], and weight sharing [25]. These strategies utilize recorded performance information to assess architectural performance directly. To enhance the search speed of NAS, Ma et al. [26] introduced a novel single-domain generalized predictor. During the training of the predictor, a feature extractor is first employed to learn domain-invariant features from pre-trained architectures. Subsequently, the architecture features are mapped to a target space to predict its accuracy. Additionally, the study proposes a regularization technique to optimize the predictor, utilizing a multi-head attention mechanism to improve its generalization capability. To address the challenge in NAS where performance predictors require multiple known architectures for training, Xiao et al. [17] proposed an ENAS framework incorporating a semi-supervised predictor. The study enhances the predictive accuracy of the semi-supervised predictor by employing a one-shot extractor and a robust regressor. Using a multi-objective optimization approach, this framework achieves a balance between prediction confidence and accuracy, leading to highly competitive architectures on benchmark test datasets. Multi-fidelity evaluation is a crucial technique to significantly reduce computational costs while maintaining the effectiveness of the search process. Traditional NAS methods often rely on fully training each candidate architecture to convergence in order to assess its performance, which is prohibitively expensive, especially when searching over large and complex design spaces. Multi-fidelity evaluation addresses this challenge by approximating the final performance of candidate architectures using cheaper, lower-fidelity proxies during the early stages of the search.

However, achieving high prediction accuracy with these performance predictors remains challenging. However, using the performance predictor to compare the relative quality of two individual architectures is highly effective. Simple machine learning methods can efficiently perform binary classification for this purpose. This inspired us to avoid directly predicting the performance of individual architectures with the performance predictor. Instead, we employ a classifier to assess the quality of two individuals and only perform the expensive true evaluation for the superior individual. This approach maximizes the utilization of expensive true evaluations, reduces the number of ineffective evaluations, and fully leverages the predictive capabilities of the performance predictor.

Numerous studies have employed performance predictors as a strategy to accelerate neural architecture search (NAS), demonstrating significant reductions in computational costs. However, most existing approaches rely on performance predictors to directly estimate architecture performance, a method that often suffers from limited accuracy. This limitation arises from two key factors:

1. Predictor Complexity: Performance predictors are typically implemented using simple machine learning models to ensure fast training and inference. While efficient, these models cannot often evaluate complex neural architectures accurately.

2. Encoding Complexity: Neural architecture encodings are high-dimensional and structurally intricate, making it difficult for predictors to extract meaningful features for precise regression-based modeling.

Prior research suggests that simple models tend to perform better on classification tasks than regression tasks, as classification only requires distinguishing between categories rather than predicting exact values. Building on this insight, we propose a novel strategy: instead of directly predicting performance metrics, our predictor classifies whether a newly generated offspring architecture is superior to its parent during the genetic algorithm's offspring generation phase. If the offspring is predicted to outperform its parent, it advances to further evaluation. This approach effectively transforms performance prediction into a binary classification problem.

We conducted an experiment to validate our hypothesis—that classification-based predictors outperform regression-based ones for neural architecture evaluation. We trained multiple neural architectures, recorded their encoded representations and CIFAR-10 accuracies, and tested two prediction approaches:

1. Regression Task: A kernel extreme learning machine (KELM) was trained to predict architecture accuracy directly. Success was measured

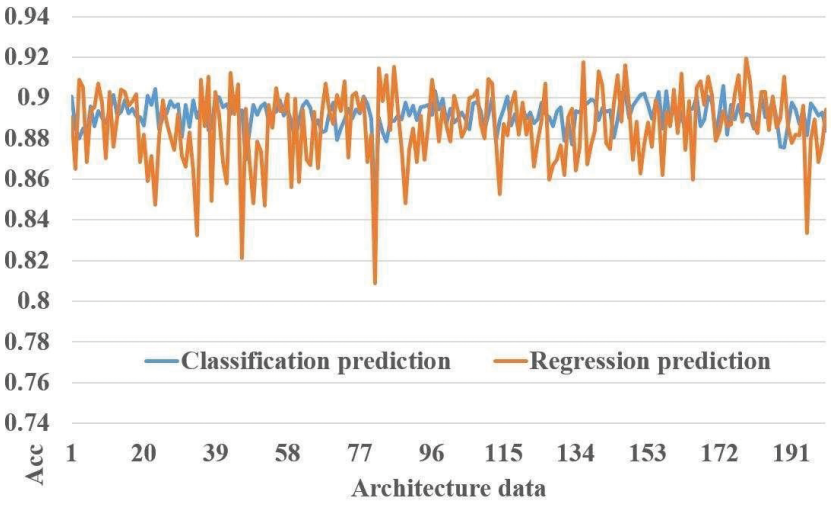

Figure 1. Comparative experiment of KELM for classification prediction and regression prediction in neural architecture data. Compared to regression prediction, KELM has higher and more stable classification accuracy for classification prediction.

by whether the predictor correctly ranked pairs of architectures based on predicted accuracies.

2. Classification Task: Paired architecture encodings were concatenated and labeled ”1” if the first architecture was more accurate, otherwise ”0”. The same KELM was then trained for binary classification.

After 200 independent trials, the results (Figure 1) clearly demonstrate the superiority of classification:

– Yellow points (regression-based prediction) show lower and less stable accuracy.

– Blue points (classification-based prediction) exhibit higher and more consistent accuracy, confirming its effectiveness in guiding architecture search.

These findings support our strategy of replacing regression with classification in performance prediction, significantly improving NAS efficiency.

This paper introduces a method to accelerate Evolutionary Neural Architecture Search (ENAS) through a combination of multi-fidelity evaluation and a genetic decision controller, aimed at reducing computational costs while maintaining high search accuracy. The multi-fidelity evaluation component focuses on reassessing individuals discarded by the evolutionary algorithm (EA). Many of these discarded individuals exhibit latent potential, and we propose reevaluating their promise using carefully designed surrogate metrics.

The primary contributions of this work are as follows:

1. We propose a genetic decision controller to accelerate ENAS. This controller evaluates the quality of offspring architectures generated during the search process. If an offspring is predicted to outperform its parent, it is subjected to a full, computationally expensive evaluation. This selective evaluation strategy significantly reduces the computational overhead of ENAS.

2. Building on multi-fidelity evaluation and the genetic decision controller, we present MFGENAS, a novel ENAS framework. Within MFGENAS, we define the search space, initialization method, and individual encoding scheme. The framework is implemented under the NSGA-II optimization framework, with dual objectives of minimizing prediction error and architectural complexity. By integrating tailored mutation operations, multi-fidelity evaluation, and the genetic decision controller, MFGENAS efficiently discovers high-performing neural architectures with minimal computational resources.

3. We employ a kernel extreme learning machine (KELM) to optimize the genetic decision controller's performance, with its hyperparameters tuned by a crisscross strategy-enhanced polar lights optimizer (PLO). The PLO is a recently proposed swarm intelligence algorithm, and we enhance its exploration capabilities through the crisscross strategy, enabling more effective optimization of the KELM's kernel parameters.
4. We conduct extensive experiments to validate MFGENAS's efficacy. Our evaluations include:
   - Benchmark comparisons against state-of-the-art methods on CIFAR-10 and CIFAR-100, where MFGENAS achieves a test error of just 2.39% on CIFAR-10 with only 0.3 GPU days.
   - Scalability tests on ImageNet, demonstrating broader applicability.
   - isualization of the MFGENAS search process and ablation studies on the genetic decision controller, confirming its critical role in efficiency gains.

The remainder of this paper is organized as follows:

- Section 2 reviews related work, including evolutionary neural architecture search and NAS acceleration techniques.
- Section 3 details the proposed MFGENAS methodology.
- Section 4 presents experimental results and analysis.
- Section 5 concludes the paper and discusses future directions.

2 Related Work

We will introduce essential background information on NAS in this section, and then discuss various research efforts that aim to lower the search costs associated with NAS.

2.1 Neural Architecture Search

NAS can be regarded as an optimization task aimed at identifying the optimal network architecture within a predefined search space. It can be defined as a single-objective optimization problem.

$$\min_x F(x) = 1 - f_{\text{valacc}}(x, D) \tag{1}$$

Let Let $D = (D_{train}, D_{val}, D_{test})$ represent the dataset, and x denote the optimal architecture found within the search space. The function $f_{valacc}(x, D)$ indicates the validation accuracy achieved by architecture x after being fully trained on D_{train} and evaluated on D_{val}.

While early NAS approaches focused solely on the performance of architectures, practical applications also necessitate consideration of training speed and architectural complexity. Consequently, smaller neural networks tend to be favored in real-world scenarios. As a result, researchers have reformulated NAS as a dual-objective optimization problem.

$$\min_x F(x) = \begin{cases} f_1(x) = 1 - f_{\text{valacc}}(x, D), \\ f_2(x) = f_{\text{complexity}}(x), \end{cases} \tag{2}$$

In Eq. (2), $f_1(x)$ represents the validation accuracy of architecture x after thorough training on D_{train} and evaluation on D_{val}. Meanwhile, $f_2(x)$ indicates the complexity of the searched architecture x. Model complexity can be quantified in several ways, typically through metrics such as floating-point operations, estimated runtime, and the number of parameters. In this study, the number of parameters is used as a direct measure of architectural complexity.

To address this multi-objective optimization problem, researchers have explored various methods, including reinforcement learning (RL) [11], gradient optimization [12], Bayesian optimization [13, 14], and EA [15, 1].

RL [27] is a machine learning paradigm that enables an agent to learn an optimal policy through interactions with its environment to maximize cumulative rewards. Unlike supervised learning, which relies on labeled data, RL learns behavioral strategies based on a trial-and-error mechanism, making it well-suited for sequential decision-making tasks. RL primarily consists of an agent and an environment. During optimization, the agent continuously interacts with the environment and receives feedback. Upon perceiving certain states from the environment, the agent selects specific actions-also

called decisions-based on certain algorithms. These actions are executed within the environment, returning the next state and a corresponding reward to the agent. This reward guides the agent to adjust so that its subsequent actions better align with the desired objective. The agent does not have prior knowledge of the optimal action at each step; instead, it must progressively identify which actions yield higher returns through ongoing interaction with the environment. Essentially, the agent's acquisition of decision-making ability is a continuous trial-and-error exploration process. Baker et al. [28] employed Q-learning-based RL in NAS. The action space includes operations such as adding layers and completing the construction of an architecture. Early-stage architectures are treated as states, and trajectories sampled from this space correspond to subsequent models that are trained to evaluate their validation accuracy. The Q-function is updated using experience replay. The study adopts an ε-greedy strategy to balance exploration and exploitation, where random trajectories are selected with a probability of 0.5. By selecting trajectories that involve multiple decision steps, the algorithm eventually reaches terminal states, at which point the corresponding models are trained, and the action-value function is updated accordingly. Negrinho et al. [29] addressed this problem using a Monte Carlo Tree Search. By adopting a tree-structured state-action space that supports incremental exploration and expansion, they utilized the upper confidence bound applied to the trees algorithm [30] to guide exploration based on the principle of optimism in the face of uncertainty. Lyu et al. [31] proposed a multi-objective RL-based NAS method that simultaneously balances accuracy, number of parameters, FLOPs, and inference latency. Combining two-stage training with pre-computation and memory-resident feature maps effectively reduces time consumption, enabling approximately 1,000 search iterations to be completed within two GPU days. To accelerate the convergence of lightweight candidate architectures, knowledge distillation was incorporated during the training phase of the search process. This also provides a more informative and stable reward signal for the RL agent, facilitating effective convergence.

Gradient-based NAS is an approach that iteratively constructs the upper or lower bounds of a function by following the direction of steepest descent or ascent. In 2018, researchers from Carnegie Mellon University proposed the first gradient-based NAS algorithm—DARTS. This method relaxes the discrete set of candidate operations into a continuous domain, using a softmax function to assign weights to all possible operations, thereby enabling a differentiable operation selection mechanism. Ultimately, the operation with the highest weight is used to construct the final neural network architecture. DARTS models the connections between nodes as a weighted sum of mixed operations and updates the operation weights through gradient descent during the search process. Compared to RL or evolutionary methods, DARTS significantly improves search efficiency, allowing the entire search to be completed on a single GPU and significantly reducing computational costs. Numerous gradient-based NAS methods have since been proposed. For example, SNAS [32] introduced a factorized distribution to represent operation selection, enabling training by activating only a single path within the supernet, thereby avoiding storing all paths in memory. Similarly, ProxylessNAS [33] adopted a parameterized distribution over operations and employed a gating mechanism for optimization, where each gate selects a path based on learned probability distributions. Xu et al. [34] proposed a partial channel connection method that randomly samples a subset of channels instead of sending all channels through the operation selection, thus reducing memory consumption. Sukthanker et al. [35] introduced a strategy to adapt gradient-based methods for weighted entangled spaces. Xie et al. [36] effectively addressed gradient estimation through integrated zeroth-order approximations, employing the sparsemax function and simulated annealing to achieve a clearer and more interpretable architecture distribution search. They also proposed a variable sized search scheme to generate compact yet accurate architectures, thereby establishing a new balance between model complexity and performance. One notable issue during supernet training is multi-model forgetting, where those of newly sampled architectures overwrite weights learned by previously trained architectures due to structural overlaps between them.

EA is an effective NAS strategy which searches for the optimal individuals through population updates, thereby alleviating the impact of gradient information and architectural complexity. These al-

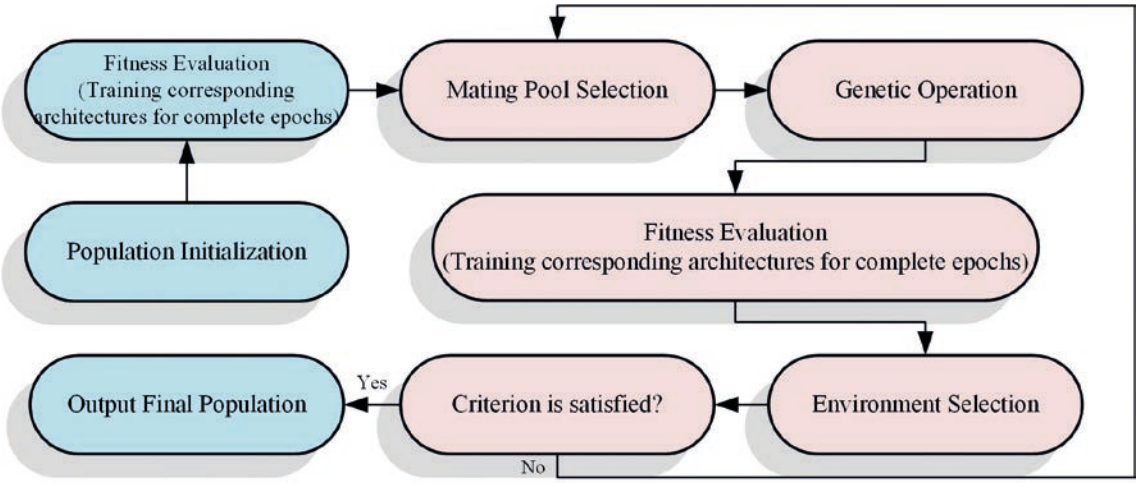

Figure 2. The framework of ENAS.

gorithms mimic the processes of biological evolution by employing mechanisms such as crossover, selection, and mutation to refine a population of candidate solutions iteratively. The process begins with an initial set of solutions, each evaluated against a predefined fitness function. The most promising solutions are then selected to produce offspring through genetic operations, enabling the creation of new generations. This cycle continues until a designated stopping criterion is reached, such as a set number of iterations or the discovery of an optimal solution. Unlike gradient-based or RL methods, EA do not require the search space to be differentiable, making them particularly suitable for exploring complex and discrete architectural configurations. In ENAS, each individual in the population encodes a candidate neural network architecture, and the optimization process mimics biological evolution through iterative application of selection, mutation, crossover, and replacement. Initially, a diverse population of architectures is randomly generated. Each architecture is partially or fully trained and evaluated on a validation set to obtain a fitness score, typically based on metrics such as accuracy, parameter count, FLOPs or inference latency. High-performing architectures are more likely to be selected for reproduction. Mutation introduces small stochastic modifications to the architectural representation, while crossover combines features from two parent architectures to create new offspring. This iterative evolutionary cycle enables the population to converge toward architectures with superior performance progressively. Importantly, EA allow explicit encoding of neural architectures, often in the form of fixed-length strings, graphs, or trees, which define both the structural topology and associated hyperparameters of the networks. This flexibility and adaptability make evolutionary algorithms compelling for efficiently navigating large and highly irregular NAS search spaces. Figure 2 shows the optimization flowchart for architecture search using EA, where the evaluation of individual fitness values is trained and validated on the dataset using complete epochs. In Figure 2, the optimization workflow for architecture search by EA is presented, highlighting that the assessment of individual fitness values involves training and validating on the dataset over full epochs.

2.2 Related Work

Numerous approaches have been developed to reduce the computational cost of NAS, which can generally be categorized into two main types: Reduce the frequency of expensive evaluations and accelerate the speed of expensive evaluations. Table 1 summarizes the acceleration methods and related literature for NAS.

1. Surrogate model: The introduction of proxy models in NAS significantly accelerates the architecture evaluation process. By leveraging performance data from existing architectures, these models can swiftly predict the performance of new designs. Common methods for constructing proxy models include Gaussian processes, random forests, and neural net-

Table 1. Methods to reduce NAS evaluation costs.

Acceleration type	Acceleration method	Literature
Reduce the frequency of expensive evaluations	Surrogate models	[26, 17, 25]
	Evaluation-free metrics	[37, 21, 1, 18]
	Population reduction and memory	[38, 15, 39]
Accelerate the speed of expensive evaluations	Early stopping	[40, 41, 42]
	Multi-fidelity	[22, 39, 43, 44]
	Subset training	[24, 45, 46]
	Weight inheritance and sharing	[23, 12, 47, 48, 49]

works. Proxy models not only reduce the time and computational resources required for evaluation but also effectively balance exploration and exploitation, optimizing the overall search process. Once the optimal architecture is identified, it typically undergoes complete training to ensure its performance in practical applications. Ma et al. [26] introduced a novel single-domain generalized predictor. During the training of the predictor, a feature extractor is first employed to learn domain-invariant features from pre-trained architectures. Subsequently, the features of the architecture are mapped to a target space to predict its accuracy. Additionally, the study proposes a regularization technique to optimize the predictor, utilizing a multi-head attention mechanism to improve its generalization capability. Xiao et al. [17] proposed an ENAS framework incorporating a semi-supervised predictor. The study enhances the predictive accuracy of the semi-supervised predictor by employing a one-shot extractor and a robust regressor. Wang et al. [25] introduced a search-free NAS method called PreNAS. This approach reduces the sample size using a zero-cost selector and enhances search speed through weight sharing. However, this method generally requires pre-training of performance predictors, which consumes a significant number of resources. Our research uses a genetic decision controller to predict the performance of the generated offspring, which can essentially be seen as a surrogate model. The difference from these traditional performance predictors is that our surrogate model mainly performs classification prediction tasks to distinguish the performance of offspring and parents, which will be more efficient than regression prediction tasks.

2. Evaluation-free metrics: To address the challenges posed by small and diverse lesion features in medical images, Wang et al. [37] proposed a training-free multi-scale NAS called MSTF-NAS, which operates on high-resolution medical images. MSTF-NAS employs a newly designed reduced search space that enables the identification of multi-scale features during the search process. During training, training-free metrics are utilized as proxy performance indicators. Lopes et al. [21] proposed a GEN method to guide the search process in NAS. The genetic evolution algorithm employs a zero-proxy estimator during the initialization phase of architecture search to evaluate the performance of individual architectures, thereby directing the evolutionary process. Following this guidance, high-scoring individuals are trained for performance and passed on to the next generation. The MOENAS method proposed by Luong et al. [1] utilizes NSGA-II to explore the search space and incorporates multiple training-free performance metrics as optimization objectives. This approach is regarded as a proxy evaluation method, as it replaces the performance assessment of individual architectures with these new training-free metrics. Phan et al. [] proposed a local search method known as potential solution improvement. By utilizing training-free metrics and improving potential solutions, the study explores architectures that balance performance and complexity within the search space. Experimental results demonstrate that compared to baseline methods, the computational cost can be reduced by a factor of four, while still enabling the exploration of certain potential solutions near the Pareto front. Evaluation free metrics are a good method for accelerating evalua-

tion. However, the selection and setting of evaluation indicators is a complex problem. The difference between evaluation-free metrics and our study is that the metrics used to evaluate the architecture's performance do not need to be trained. However, the genetic decision controller proposed in our study also does not require much training data. Evaluation-free metrics usually use information such as activation functions in neural networks to judge the architecture's performance, and the accuracy of such predictions is not very high.

3. Population reduction and memory: Population reduction accelerates the search process by effectively decreasing the number of architectures that need to be evaluated, thereby reducing computational overhead. This is achieved by establishing appropriate selection criteria to identify high-performing and diverse architectures for further evaluation. Concurrently, the population memory mechanism retains the characteristics of historically well-performing architectures, helping to avoid the redundant assessment of inefficient designs. This memory capability not only speeds up the search process but also facilitates the effective exploration of new architectures, enhancing both the comprehensiveness and depth of the search. Fan et al. [38] categorized the optimization process of the ENAS method into different stages with varying population sizes. They utilized larger populations to enhance the global search capability while employing smaller populations to reduce the number of individual architectures evaluated within the population. The CMANAS method proposed by Sinha et al. [15] incorporates a covariance matrix adaptation evolution strategy into the NAS search process, achieving improved search results. In this study, rather than retraining each individual architecture, a one-shot model is used to validate accuracy. An adaptation table is maintained to record the fitness values of evaluated individuals, allowing for direct retrieval of fitness values when the same architecture is encountered again. The problem with population reduction and memory is that it reduces the accuracy of architecture evaluation. When the population diversity is high, the effectiveness of this method will weaken. Population reduction and memory are difficult to perform well when the population is diverse. However, the genetic decision controller proposed in our study can work well for populations of different diversities.

4. Early stopping: Introducing early stopping mechanisms has significantly enhanced search efficiency. This approach establishes specific criteria for early stopping, such as monitoring loss or accuracy on the validation set, to determine when to halt the training of underperforming architectures dynamically. Specifically, after each training epoch, the model's validation performance is assessed, and if no significant improvement is observed over a predetermined number of epochs, the training of that architecture is promptly terminated. This method not only reduces the waste of computational resources but also accelerates the identification of potentially high-performing architectures. Suganuma et al. [40] updated the reference curve based on the best individual from each iteration and designed an early stopping strategy using this reference curve to accelerate ENAS. Early stopping can lead to a decrease in evaluation accuracy, and when to stop is a challenge. The early stopping strategy can be regarded as a low-fidelity evaluation strategy. In our study, we also use some custom indicators as a multi-fidelity evaluation method to evaluate the neural architecture.

5. Multi-fidelity: Multi-fidelity evaluation usually uses multiple indicators to evaluate architecture performance. These indicators have different fidelity, and better performance evaluation results can be achieved through effective combination. Liu et al. [22] reduce the time consumption during architecture evaluation through multi-fidelity training. The study employs low-fidelity assessments as the primary objective while utilizing high-fidelity evaluations to train the performance predictor of the proxy model. By combining parameter sharing, performance prediction, and low-fidelity evaluations, the search space is explored effectively. Since low-fidelity assessments require fewer evaluations, they can also be considered a form of early stopping mechanism. However, while early stopping strategies can expe-

dite ENAS, they may lead to inaccuracies in evaluation, particularly when dealing with more complex architectures, making it challenging for early-stopping-based ENAS to achieve optimal results [39]. Trofimov et al. [43] introduced a Bayesian multi-fidelity approach to enhance the search speed of NAS. In this study, knowledge distillation is employed for training over several epochs, allowing individual architectures to learn from the performance of a teacher network, thereby reducing training time. Sun et al. [44] proposed a multi-fidelity evaluation NAS method called ES-CRNA-ME. This approach utilizes weight sharing techniques to search for robust architectures, incorporating semantic adversarial attacks, various norm-based attacks, and composite adversarial attacks. Additionally, ES-CRNA-ME employs a multi-fidelity online surrogate model to further reduce search costs.

6. Subset training: The application of subset training strategies significantly accelerates the speed and efficiency of architecture evaluation. This approach involves preliminary training on a smaller data subset to assess the potential performance of architectures quickly. By effectively reducing the training time for each architecture, this method enables faster identification of high-performing architectures within limited computational resources. Gao et al. [24] introduced an innovative heterogeneous graph NAS method called HGNAS, designed to automatically search graph neural networks. HGNAS trains an RNN controller using a generative adversarial learning framework to enhance search speed and incorporates a pairwise ranker into the RL process as a search strategy. During the search, only a small subset of architectures is evaluated using the pairwise ranker, which accelerates the overall search process. Liu et al. [45] enhance search speed by loading subnetworks into a pre-trained super-net and employing federated learning to parallelize the search for optimal network architectures. Wang et al. [46] introduced a parallel differential NAS method called FP-DARTS. This approach utilizes a super-net to accelerate the search process and is designed from three levels. First, the search space is divided into two subsets, each containing four operations, and two subnetworks are used to search within each subset, constructing a parallel super-net. Second, a limited connection strategy is employed at the channel level, avoiding connections for all channels. Finally, a parameter is used to control whether a subnetwork participates in the super-net training. Subset training does not guarantee the accuracy of training, and choosing a dataset is a critical issue.

7. Weight inheritance and weight sharing: Weight inheritance allows new architectures to borrow weights from previously trained models during the training process, thereby accelerating convergence and reducing computational resource consumption. This approach not only speeds up the training of new architectures but also allows for the inheritance of performance advantages from the prior models. Concurrently, the weight sharing strategy enables multiple architectures to share the same parameters within a single model, further lowering computational costs. This sharing mechanism enhances the efficiency of performance evaluation across different architectures, as multiple trainings can occur in parallel, avoiding redundant calculations. By combining weight inheritance and sharing, NAS can explore high-performance network architectures in a shorter timeframe, effectively improving both the efficiency and quality of model design. Wang et al. [23] introduced a collaborative multi-task ENAS approach for image classification in remote sensing scenes. This method leverages the similarities among tasks in different image datasets to facilitate transfer learning. Experimental results validate the effectiveness of the proposed adaptive collaborative transfer approach. To address non-differentiable tasks such as resource constraints, energy efficiency, and other non-differentiable metrics. Lyu et al. [12] proposed a differentiable NAS approach. This method incorporates non-differentiable evaluation metrics and eliminates the need for training a sampling controller. To enhance the robustness of NAS architecture searches. Chen et al. [47] constructed a NAS framework using multiple teacher-guided approaches. The study employs adaptive ensemble methods and disturbance-aware knowledge distillation algorithms. Zhang et al. [48] intro-

duced NASRec to search for neural architectures applicable to recommendation systems. Before the search begins, a large super-net is constructed to serve as the search space, and during the subsequent search, individual architectures inherit the weights and performance evaluations from the super-net. Wang et al. [25] introduced a search-free NAS method called PreNAS. This approach reduces the sample size using a zero-cost selector and enhances search speed through weight sharing. Peng et al. [49] proposed a one-shot architecture search framework called RSB-Net, which is based on EA and weight sharing strategies. This method first trains a super-net within a hierarchical search space and then conducts a search for neural architectures suitable for remote sensing image classification using a zero-training approach. Weight inheritance and weight sharing have a limited role when diversity within populations is well established. Our proposed NAS method can perform well when the population diversity is high.

3 The Proposed Approach

In this section, we will introduce the proposed ENAS method, MFGENAS, which utilizes multi-fidelity evaluation based on genetic decision controller. In particular, this section will detail the overall structure of MFGENAS, the encoding methods designed, the crossover and mutation techniques, and the introduced multi-fidelity evaluation approach based on genetic decision controller.

3.1 Overall Framework of MFGENAS

In this section, we will describe the overall framework of the proposed MFGENAS. To concurrently optimize the accuracy and complexity of the networks, we utilize NSGA-II as the optimizer. To expedite the search process, we train the accuracy of individual architectures using varying epoch training as multiple fidelity evaluation indicators. Additionally, we apply an enhanced swarm intelligence algorithm, CPLO, to optimize the hyperparameters of the KELM, which acts as the genetic decision controller in the multi-fidelity evaluation. Figure 3 depicts the flowchart of MFGENAS, which is mainly divided into population initialization, architecture evaluation, selection, crossover and mutation through genetic operators, and multi-fidelity evaluation based on a genetic decision controller.

Initially, we create a population through the initialization process. During this stage, we modify the encoding of individuals so that some generated architectures resemble ResNet or Inception networks. We then evaluate the fitness values of the initialized population. This evaluation method consists of training and validating these individuals on the dataset for 1 epoch to obtain validation accuracy, using images such as CIFAR-10 as the dataset for the experiments.

Next, we utilize a binary selection method to choose certain individuals from the population for the subsequent crossover and mutation operations. We generate a descendant population from two parent individuals through the designed crossover and mutation operator.

We use CPLO optimized KELM as a genetic decision controller to determine whether the generated offspring are effective. KELM will determine whether the generated offspring are superior to the parent. If the offspring are superior to the parent, proceed to the next step; otherwise, regenerate the offspring. The resulting population undergoes training and validation on the dataset to obtain validation accuracy, with the training epochs adaptively updated according to the search iteration count, beginning with 1 epoch and reaching a maximum epoch equal to the predefined maximum fidelity value. Lastly, the multi-fidelity evaluation method is used to select the best individuals from the combined population of parent and descendant individuals. Importantly, this method identifies overlooked superior individuals among those eliminated by NSGA-II, thus preventing resource wastage during evaluations.

Algorithm 1 outlines the detailed steps of MFGENAS. The MFGENAS is designed to efficiently explore and optimize neural network architectures through a multi-fidelity approach and a genetic decision controller. The algorithm begins by initializing a population X based on a specified range of node counts R_{node} and a defined population size N. E is a repository that stores the individual architecture information used to store potentially promising individuals selected by NSGA-II. S represents the number of epochs in each iteration.

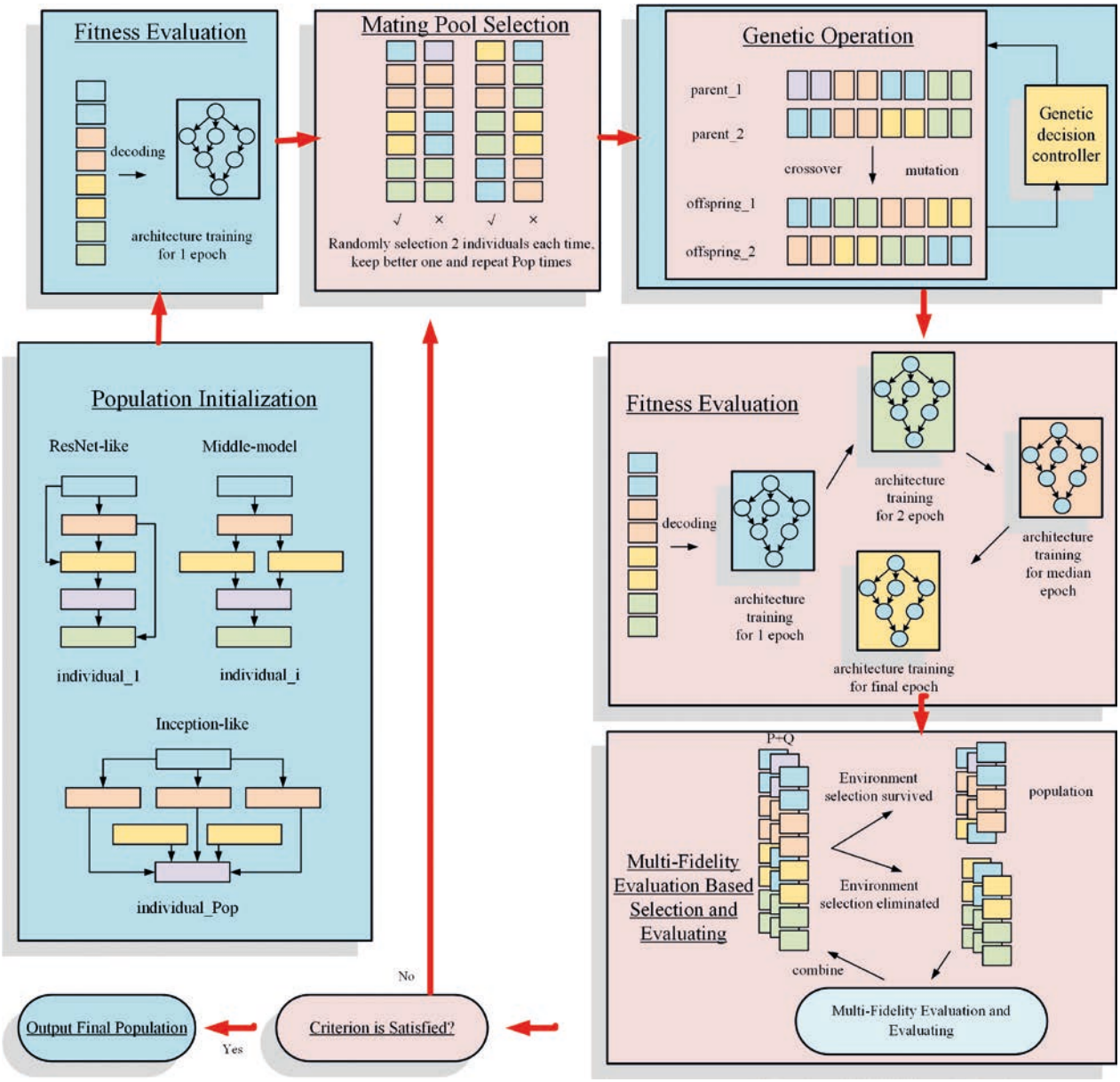

Figure 3. The main process of MFGENAS. (Mainly divided into initialization, fitness value evaluation selection, crossover mutation, genetic decision controller, and multi fidelity evaluation selection.

Each individual in the initial population X is trained for S epochs to compute their fitness values. The algorithm's core consists of a loop that iterates from 1 to the maximum number of evaluations *MaxFes*. In each iteration, $N/2$ parent individuals are selected from the current population x to form a subset denoted as X'. Genetic manipulation techniques are then applied to X' to produce offspring X''.

Algorithm 1. Main processes of MFGENAS

Input: MaxFes : Maximum number of searches; NF : Fidelity N : Population size; R_{node}: Initialize a range of node counts;
Output: X: Population;
X, E, S←Initialize populations based on R_{node} and N;
The individuals in X are trained for S epochs to obtain their fitness values;
For Fes = 1 to MaxFes:
  X′ ←Select N/2 parent individuals in X;
  For x′ in X′:
    x″ ←Offspring are generated based on x′;
    Timer = 0
    While whether x″ is inferior to x′ determine by genetic decision controller and Timer < 10:
      x″ ←Offspring are generated based on x′;
        Timer←Timer+1;
    End while
    X″←x″;
  End For
  The individuals in X″ are trained for S epochs to obtain their fitness values;
  X, E, S←Selection is performed using multi-fidelity evaluation; % Algorithm 2
End For
Return X

When generating offspring, each offspring generated by the parent is judged by a genetic decision controller to determine its validity. If the genetic decision controller determines that the generated offspring is inferior to the corresponding parent, the offspring will be regenerated. If the genetic decision controller determines that the generated offspring are inferior to the corresponding parent offspring, the generated offspring will be retained. This process lasts ten times. Following this, individuals in X'' are trained for S epochs to obtain their fitness scores.

The selection process for the next generation implements a multi-fidelity evaluation strategy to update the populations X, E, and S based on the evaluated fitness. This process continues until the termination criteria are met, ultimately returning the optimized population X, which encompasses the most promising neural architectures identified throughout the search process.

3.2 Encoding Schemes and Genetic Operations

In this section, we will discuss the search space and encoding methods used in MFGENAS. The search space is based on a cell architecture. An individual network comprises multiple cells, which include convolution cells (referred to as CC) and pooling cells (referred to as PC).

Each cell can be viewed as a mapping from a tensor of dimensions $H \times W \times F$ to a tensor of dimensions $H' \times W' \times F'$. Convolution cells maintain the original size of the input tensor during mapping, whereas pooling cells have mappings defined as $H' = \frac{1}{2}H$, $W' = \frac{1}{2}W$, and $F' = \frac{1}{2}F$. Figure 4 shows examples of both CC and PC, where the CC consists of 10 nodes and the PC consists of 6 nodes. Each cell contains two input nodes, one output node, and one connection node, with the input nodes receiving outputs from the parent cell and the grandparent cell, respectively.

We employ a strategy for each node that involves summing the inputs before performing operations. The choices of operations are documented in Table 2, which consists of four convolution operations and six pooling operations.

In MFGENAS, an individual architecture includes two cell types: CC and PC. Fig. 4 provides an example of how an individual architecture is encoded. Specifically, an individual architecture can be formulated as shown in Eq. (3).

$$\begin{cases} \text{CC} = (n_0, n_1, \ldots, n_k, \ldots, n_{N-1}), & k = 0, 1, \ldots, N-1 \\ \text{PC} = (p_0, p_1, \ldots, p_m, \ldots, p_{R-1}), & m = 0, 1, \ldots, R-1 \end{cases} \tag{3}$$

In this context, n_k denotes the $k-th$ node within the convolution cell, and p_m signifies the m-th node within the pooling cell. N represents the total number of nodes in the convolution cell, while R refers to the total number of nodes in the pooling cell. Both n_k and p_m are represented as subvectors, which include link operations for the pre-

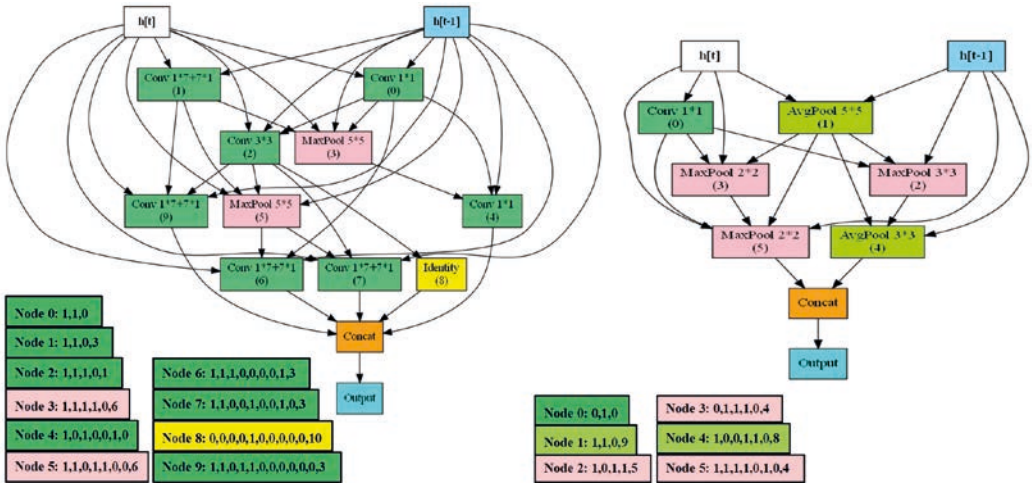

Figure 4. An individual architecture can be illustrated as individual = (CC, PC).

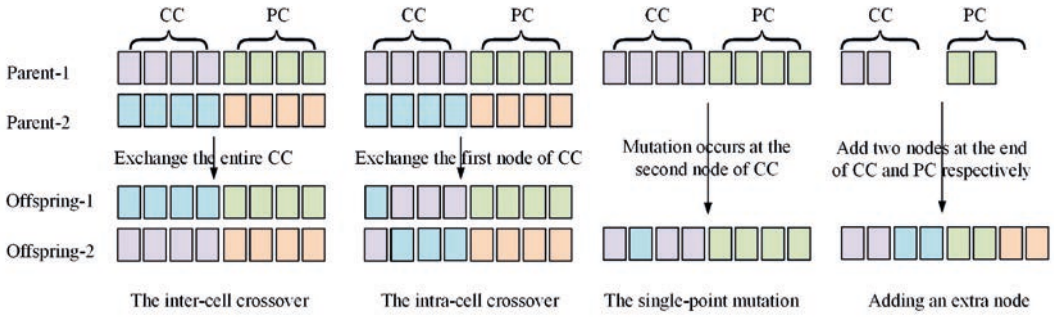

Figure 5. Four types of genetic operations.

ceding node as well as for the input. Specifically, we can consider n_k as an example.

$$\begin{aligned} n_k &= (L, O_P) = (\hat{l}_0, \hat{l}_1, l_0, l_1, \ldots, l_i, \ldots, l_{k-1}, O_P), \\ &\hat{l}_0, \hat{l}_1 \in (0,1), \\ &i = 0, 1, \ldots, k-1, \\ &l_i \in (0,1), \quad O_P \in \{0, 1, \ldots, 10\} \end{aligned} \tag{4}$$

In this case, $L = (\hat{l}_0, \hat{l}_1, l_0, l_1, \ldots, l_i, \ldots, l_{k-1})$ represents the link vector associated with node n_k, where $\hat{l}_0, \hat{l}_1$ indicate whether there are links to the first and second input cells, respectively. l_i indicates whether there is a link to the i-th node, while O_P denotes the operation type selected for the node, as detailed in Table 2.

To provide clarity on the encoding method for individual architectures, we focus on node 2 from the CC in Figure 4. The encoding for node 2 is [1,1,1,0,1], which shows that node 2 is connected to the first input, connected to the second input, linked to node 0, not linked to node 1, and has an operation type of 1, representing a convolution with a kernel size of 1×1.

Table 2. Search space.

Operation	Number
Identity mapping	0
Convolution with kernel size 1×1	1
Convolution with kernel size 3×3	2
Convolution with kernel size 1×3	3
Convolution with kernel size 1×7	4
Max pooling with kernel size 2×2	5
Max pooling with kernel size 3×3	6
Max pooling with kernel size 5×5	7
Average pooling with kernel size 2×2	8
Average pooling with kernel size 3×3	9
Average pooling with kernel size 5×5	10

We have designed crossover and mutation methods tailored to the proposed encoding scheme as genetic operations to achieve our optimization objectives. As illustrated in Figure 5, we have identified four types of genetic operations: inter-cell crossover, intra-cell crossover, single-point mutation, and adding an extra node. In the case of inter-cell crossover, CCs or PCs from two parent individuals are randomly exchanged to create new offspring. For intra-cell crossover, a random position is chosen within the two cells to swap nodes from each parent. If the selected position exceeds the length of one of the cells, the excess nodes are appended to the shorter cell. In single-point mutation, a random node undergoes mutation, altering only the operation type, randomly selected from the search space. Finally, for adding an extra node, a random node is added to the end of either the CC or PC, resulting in a new individual.

3.3 Multi-Fidelity Evaluation

In this section, we discuss the proposed multi-fidelity evaluation approach. We substantially cut down the evaluation time by minimizing the number of epochs required for evaluations, thus accelerating the search process. Furthermore, since individual evaluations can be time-intensive, valuing the results more carefully is important. Normally, individuals eliminated by NSGA-II are simply discarded, which is an inefficient use of evaluation resources. Consequently, our multi-fidelity evaluation primarily concerns those individuals eliminated by NSGA-II. Among these discarded individuals, many possess promising potential, and we seek to reassess their worth through several specially designed non-evaluative metrics.

To effectively monitor individual statuses and select non-evaluative metrics, we have developed three markers: SSC, MSC, and MEC. Specifically, SSC denotes the count of surviving according to one evaluation, MSC denotes the count of surviving according to multi-fidelity evaluation, and MEC denotes the count of being eliminated according to multi-fidelity evaluation.

Algorithm 2 presents the specific process for the multi-fidelity evaluation method. The algorithm's inputs consist of the population X, the offspring X'', and a repository E designated for promising individuals. Additionally, it requires the current iteration count, maximum iteration count, fidelity NF, and the current epoch S. NSGA-II is then used to select N individuals from both the population X and the offspring X''. The selected optimal individuals are marked as X_S, while those that are eliminated are referred to as X_E. The multi-fidelity evaluation is subsequently applied to identify potential candidates within X_E. As a result, the SSC variable for individuals in X_S is increased by 1, while the

MEC variable for individuals in X_E is also incremented by 1.

Algorithm 2. Main processes of multi-fidelity evaluation

Input: X: population; X′′: offspring; E:Archive; Fes:Current number of iterations; NF: fidelity; S: Current number of training epochs; MaxFes: Maximum number of searches;
Output: X: Population; E: Archive; S: Current number of training epochs;

1. X_S , $X_E \leftarrow$ N individuals were selected in X+X′′by NSGA-II;
2. SSC + 1 of each individual in X_S;
3. MEC + 1 of each individual in X_E;
4. T ←Individuals in $E + X_E$ are ranked using three designed non-evaluative metrics;
5. E←Select the top half of the individuals from T;
6. **IF** $\mathrm{mod}\left(\mathrm{Fes}, \left\lfloor \frac{\mathrm{MaxFes}}{\mathrm{NF}} \right\rfloor\right)$==0 and S≠NF:
7. S = S + 1;
8. Train the individuals in $X_S + E$ for S epochs to obtain their fitness values;
9. X, E←Select N individuals from $X_S + E$ using NSGA-II;
10. MSC +1, SSC =0 in X;
11. MSC + 1 in E whose MEC == 0;
12. MEC + 1 in E;
13. **ELSE IF:**
14. X←X_S;
15. **END IF**
16. **IF** FES = MaxFes :
17. Train the individuals in $X_S + E$ for a complete epoch to obtain their fitness values;
18. X, E←Select N individuals from $X_S + E$ using NSGA-II;
19. **END IF**

Return X, E, S

Then we rank individuals in $E + X_E$ using the three designed non-evaluative metrics. The three non-evaluative metrics are derived from *SSC*, *MSC*, and *MEC*. Initially, individuals are selected based on $MSC \cdot MEC$, which aims to keep individuals that regularly survive through multi-fidelity evaluations. Following this, the second selection criterion employs $-(MSC + MEC)$, favoring individuals with limited experience in varied fidelity evaluations. Lastly, individuals are selected based on *SSC*, which favors those who often survive based on a singular evaluation. After these evaluations utilizing non-evaluative methods, individuals are ranked to create T, and the top half are moved to the repository E.

To utilize multi-fidelity for training individual architectures, we periodically adjust the epoch count S. The frequency of these updates is governed by $MaxFes$ and NF. The specified multi-fidelity NF determines how many epoch types will be utilized in the search process. Thus, if an update is warranted during the current iteration, we increment S by 1. As S is updated, we proceed to train individuals in $X_S + E$ for S epochs to derive their fitness values. We then select individuals from $X_S + E$ using NSGA-II. At this stage, the metrics for the individuals have altered; individuals in X will have their MSC increased by 1 and SSC reset to 0, while individuals in E that have MEC=0 will see their MSC incremented by 1 and their MEC also increased by 1. If the current iteration count does not necessitate an update to S, we will simply integrate X_S into X.

If Fes reaches the maximum iteration threshold $MaxFes$, we will then conduct a full epoch of training for the individuals in $X_S + E$ to ascertain their fitness values. Following this, we will select individuals from $X_S + E$ by NSGA-II.

3.4 Genetic Decision Controller

ENAS is highly time-consuming; in addition to employing multi-fidelity evaluations to enhance search speed, we further accelerate ENAS using a genetic decision controller. Based on previous experience, simple machine learning models or shallow neural networks tend to perform better on classification tasks than regression tasks. Classification only requires determining the category to which a sample belongs rather than providing an exact numerical prediction. Inspired by this observation, we propose a novel strategy in which the performance predictor is used during the offspring generation phase of the genetic algorithm to classify whether a newly generated offspring architecture is superior to its parent. If the offspring is predicted to outperform the parent, it proceeds to the next evaluation stage. Essentially, this is formulated as a binary classification problem determining which of the two architectures performs better. We utilize a KELM optimized by an enhanced PLO for the genetic decision controller. PLO is a novel swarm intelligence optimization algorithm, which we further refine to optimize the kernel parameters of KELM.

KELM is an enhanced version of the extreme learning machine (ELM), integrating kernel methods to improve its capability in handling nonlinear data. Unlike traditional ELM, which randomly assigns weights and biases in the hidden layer and requires analytical solutions, KELM incorporates kernel functions to project data into higher-dimensional feature spaces, achieving superior classification and regression accuracy. The use of kernel functions allows KELM to bypass the need for manual feature engineering, making it suitable for complex datasets with intricate structures. KELM's architecture enables fast training and testing times, a key advantage over other machine learning methods that rely on iterative processes. Moreover, by adjusting kernel parameters, KELM can balance generalization and model complexity, providing a robust solution for high-dimensional data problems in various applications, such as image recognition, bioinformatics, and time series prediction. These attributes make KELM particularly well-suited for performance prediction tasks within NAS frameworks, where computational efficiency is crucial.

When utilizing KELM as genetic decision controller, certain hyperparameter configurations are critical to their predictive accuracy, including the kernel parameters of KLEM. To improve the prediction accuracy of the KELM as a genetic decision controller, we additionally implement an enhanced PLO algorithm to fine-tune its hyperparameters.

PLO is a swarm intelligence algorithm inspired by the aurora phenomenon. The Polar Lights Optimizer (PLO) selection is motivated by its status as a novel swarm intelligence algorithm with promising optimization performance. PLO emulates the dynamic behavior of particles subjected to forces, displacement, and aggregation within a magnetic field, thereby enabling efficient solution space exploration. PLO offers several notable advantages compared to conventional swarm-based approaches such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). First, it achieves a balance between global and local search capabilities. The "aurora mechanism" allows particles to efficiently converge around dominant poles while maintaining stochastic perturbations to enhance global exploration. Second, PLO exhibits faster convergence rates, benefiting from its direction-guided search strategy and dynamically updated pole positions, which allow it to approach optimal solutions within fewer iterations. Finally, PLO demonstrates strong robustness, as it is insensitive to initial parameter settings and is well-suited for high-dimensional, complex, nonlinear, and gradient-free optimization problems. PLO proposes a rotational movement mechanism for charged particles, integrating their velocity, trajectory, energy, and atmospheric properties to devise a strategy for laser elliptical walking. Ultimately, the particle collision phenomenon informs a strategy for particle interactions within the population, which can be expressed using Eq. (5).

$$\mathbf{v}(t) = Ce^{\frac{qB-a}{m}t} \tag{5}$$

Here, is a constant, stands for a charge, signifies the Earth's magnetic field, represents the mass of a charged particle, denotes time, and is the damping coefficient, which ranges from [1,1.5]. The formula used to produce new ions during each iteration is expressed in Eq. (6).

$$X_{new}(i,j) = X(i,j) + r_2 \times (W_1 \times \mathbf{v}(t) + W_2 \times A_0) \tag{6}$$

In this formulation, $X_{\text{new}}(i,j)$ represents the j-th dimension of the newly created i-th particle, whereas $X(i,j)$ denotes the j-th dimension of the i-th particle from the current population. The value of r_2 is constrained within the range of $[0,1]$. The two adaptive parameters, W_1 and W_2, are defined by Eq. (7) and Eq. (8), respectively.

$$W_1 = \frac{2}{\left(1+e^{-2(t/T)^4}\right)} - 1 \tag{7}$$

$$W_2 = e^{-(2t/T)^3} \tag{8}$$

where t is the current iteration count, and T is the total iteration count. A_0 denotes the intricate changes of the auroral ellipse, represented mathematically by Eq. (9).

$$A_0 = |d|^{-1-\beta} \times \left(\frac{1}{N}\sum_{i=1}^{N} X_i - X_{i,j}\right) + LB + \frac{r_1 \times (UB-LB)}{2} \tag{9}$$

where d refers to the step size, β serves as a stability adjustment metric, while the value of r_1 is within the interval [0,1].

PLO is a highly effective swarm intelligence optimization algorithm. Nevertheless, in the context of optimizing KELM parameters, we seek to enhance its algorithmic capabilities to discover superior solutions. To achieve this, we incorporate a crisscross strategy into PLO, thereby improving its development potential.

The crossover mechanism is inspired by the crossover optimizer (CSO) developed by Meng in 2014 [35]. This approach utilizes two distinct operations: vertical crossover search (VCS) and horizontal crossover search (HCS), which enable the exchange of information between particles along horizontal and vertical directions, respectively. The fundamental principle of this strategy involves generating new particles by sharing information between randomly chosen dimensions or particles. The optimal particles are retained and integrated into the population, strengthening the crossover mechanism and enhancing the algorithm's global search efficiency.

HCS involves performing crossover operations on the dimensions of two randomly chosen particles. This process enhances the utilization of population information, refines the search process, and bolsters the algorithm's global exploration capability. The HCS operation is formulated through Eq. (10) and Eq. (11).

$$HCS(i,j) = r_1 \times X(i,j) + (1-r_1) \times X(k,j) + c_1 \times (X(i,j) - X(k,j)) \quad (10)$$

$$HCS(k,j) = r_2 \times X(k,j) + (1-r_2) \times X(i,j) + c_2 \times (X(k,j) - X(i,j)) \quad (11)$$

where r_1 and r_2 are random numbers in the range [0,1], while c_1 and c_2 are within [-1,1]. The value $X(i,j)$ represents the j-th dimension of the i-th particle in the population X, and $X(k,j)$ denotes the j-th dimension of the k-th particle in the same population. K is randomly generated within the population size. The new offspring, $HCS(i,j)$ and $HCS(k,j)$, are generated through the HCS operation. Following this operation, the offspring compete with their parent particles, with those exhibiting superior fitness retained in the population.

The VCS operation is conducted on each particle by randomly selecting two pairs of dimensions for crossover to produce a new particle. As with HCS, the offspring generated by the VCS operation compete with their parent particles, with the fitter particles retained in the population. The VCS operation is defined by Eq. (12).

$$VCS(i,j) = r_3 \times X(i,j_1) + (1-r_3) \times X(i,j_2) \quad (12)$$

where r_3 is a random number within the range [0,1], while $X(i,j_1)$ and $X(i,j_2)$ represent the values of the two dimensions randomly selected by the i-th particle. The notation $VCS(i,j)$ denotes the value of the j-th dimension generated from these two randomly chosen dimensions of the i-th particle.

Algorithm 3. Main processes of CLPO optimizing KELM.

Input: Data: The architecture consists of 260 individuals designated for training and predicting with the KELM; MaxFes: Maximum number of searches; N: Population size;
Output: X_{best}: Optimal kernel parameters of KELM

```
X←Initialize populations based on N;
f(X)←Evaluate each individual in X on Data by KELM;
X_best←Update the current optimal solution;
For Fes = 1 to MaxFes do:
  v(t), Ao, W1, W2 ←Calculate the velocity, aurora oval walk and two weights;
  For n = 1 to each energetic particle do:
    X_new←Updating particles;
    If r4 < K and r5 < 0.05 :
      X_new←Particle collision strategy;
    End If
    f(X_new)←Evaluate each individual in X_new on Data by KELM;
    Fes = Fes + 1;
  End For
  If f(X_new) < f(X ) :
    X←Iterating over X using the greedy selection mechanism;
  End If
  X←Update X by crossover mechanism;
  X_best←Update the current optimal solution;
End For
Return X_best
```

Algorithm 3 and Figure 6 illustrate how CPLO optimizes KELM. In this optimization process, the predictive accuracy of the KELM on the individual architecture dataset is used as the fitness value to discover the optimal parameters for the KELM. Algorithm 1 details the method based on CLPO for

this parameter optimization. The algorithm takes as input 260 samples for training and prediction, the maximum search iterations (MaxFes), and the population size (N). Initially, a particle swarm X is created based on the specified population size, with the fitness $f(X)$ of each individual calculated using the KELM. The current optimal solution X_{best} is then updated. In the main iteration loop, the algorithm can execute up to MaxFes iterations. In each iteration, the velocity $v(t)$ of the particles, the auroral elliptical path A_o, and the two weights W_1 and W_2 are determined. Next, each energetic particle updates its position to X_{new}. If the random variable r_4 is less than the threshold K and r_5 is below 0.05, the particle collision strategy is employed to refine X_{new}. r_4 and $r_5 \in (0,1)$. The fitness $f(X_{\text{new}})$ of the newly generated particle is then re-evaluated, and the iteration count is incremented. If the new fitness value exceeds that of the current best solution, the algorithm optimizes the particle swarm through a greedy selection process. Finally, the particle swarm X is updated by the crossover mechanism, and the optimal solution X_{best} is continually improved. The algorithm ultimately outputs the best KELM parameters, effectively enhancing the accuracy and generalization performance of the KELM model.

4 Experimental Study

In this section, we will conduct experiments to assess the effectiveness of the proposed MFGENAS. We will begin by detailing the datasets utilized, the competitors in the field, and the configurations of the experimental parameters. Following this, a comparison will be made between MFGENAS and the competing methods. Furthermore, we will analyze the search process of MFGENAS, the transferability of the architectures obtained, the efficacy of the multi-fidelity evaluation, and the overall effectiveness of the genetic decision controller.

4.1 Experimental Settings

This section presents the datasets, comparison algorithms, and parameter configurations utilized in the experiments. For the datasets, we employed the CIFAR-10 dataset during the search process of MFGENAS. The optimal architectures identified were also tested for performance on CIFAR-100 [] and ILSVRC 2012 ImageNet. CIFAR-10 consists of 50,000 training images and 10,000 testing images, divided into 10 categories, with each image measuring 32×32 pixels. CIFAR-100 is similar to CIFAR-10 but expands the number of categories to 100. ImageNet is a larger dataset, containing 1.28 million training images and 50,000 testing images, with image dimensions of 224×224 pixels and 1,000 classes. During the search on CIFAR-10, the training dataset was split into 90% for training and 10% for validation.

To assess the effectiveness and efficiency of MFGENAS, we selected several state-of-the-art competing algorithms for comparison. The first category of competitors includes classical deep neural networks that have been manually de-signed, such as Wide ResNet [51], Inception-v1 [52], DenseNet [8], ShuffleNet [53]. The second category encompasses NAS methods that do not rely on EA, including NASNet [10], Block-QNN-S [54], MetaQNN [28], E-NAS [55], EAS [56], DARTS [57], NAO [58], DDP-Net [59], SQNAS [60], VNAS [61], AER[62]. The third category features evolutionary computation-based ENAS methods, such as AmoebaNet [63], hierarchical evolution [64], large-scale evolution [65], CNNGA [66], Genetic-CNN [67], NSGA-Net [68], E2EPP [69], AE-CNN [70], E2NAS-MIG [71], SI-ENAS [72] CARSE [73], LEM-ONADE [74], CARS-E [73], GENAS[75], MOEA-PS [76], and ACE [77]. Furthermore, we compared MFGENAS without the genetic deci-sion controller-based multi-fidelity evaluation, referred to as MFGENAS (base), to highlight the role of the genetic decision controller-based mul-ti-fidelity evaluation. In the search process of MFGENAS, we set the following parameters: each individual architecture was configured with 5 cells, and each node had a channel count of 16. The learning rate, momentum rate, and batch size were set at 0.1, 0.9, and 128, respectively. The population size for MFGENAS was configured to 20, with a maximum iteration count of 25 and a node range of [5, 12]. The multi-fidelity parameter was set to 6. The search was conducted using an NVID-IA 2080TI GPU. After identifying the optimal architecture, training on CIFAR-100 was execut-ed on a Tesla P100 GPU, while training on ImageNet was carried out on two Tesla P100 GPUs. For optimizing the KELM parameters using CPLO, the population size was established at 30, with a maximum iteration count of

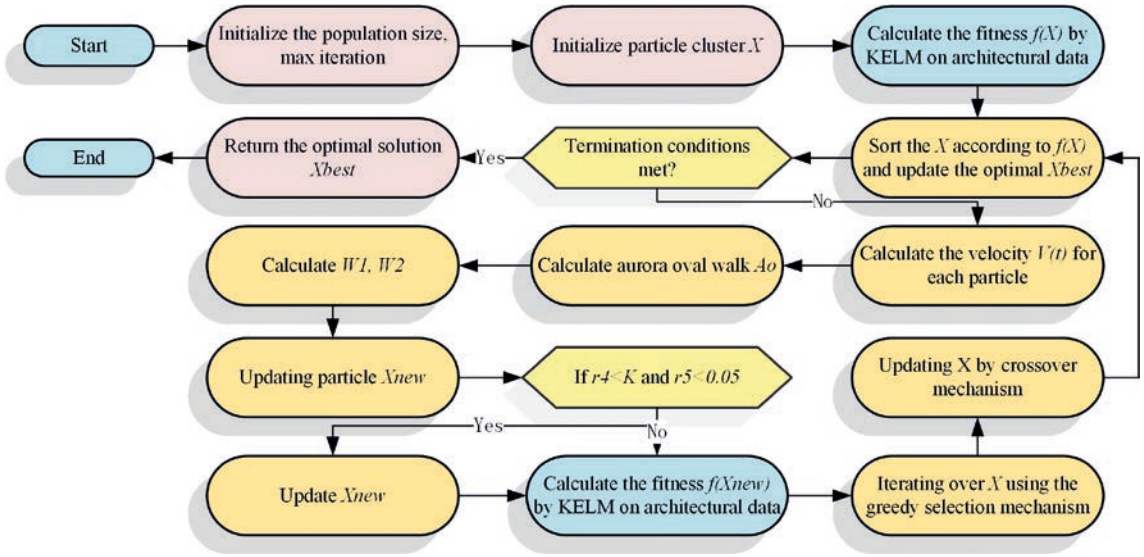

Figure 6. Main processes of CLPO optimizing KELM.

30,000. The parameters for the PLO adhering to the pa-rameters defined in the original PLO paper.

4.2 Effectiveness of MFGENAS

This section aims to demonstrate the effectiveness of the proposed MFGENAS method through comparative experiments with other competitors. The experimental outcomes for MFGENAS and other similar algorithms on the CIFAR-10 dataset are presented in Table 3. The metrics analyzed include test error (%), parameters (M), and search cost (GPU days). test error is used to indicate the performance of the discovered neural networks, while search cost reflects the efficiency of MFGENAS.

Notably, MFGENAS achieved a test error of 2.39%, the lowest among all evaluated algorithms, showcasing its exceptional performance in image classification tasks. When compared with other similarly performing models such as NASNet-A (2.65%) and DDPNAS (2.44%), MFGENAS achieves a high accuracy with just 2.94M parameters, demonstrating effective parameter utilization. Furthermore, the search cost for MFGENAS is only 0.3 GPU days, which is significantly lower than most algorithms, including NASNet-A (2000 GPU days) and AmoebaNet-A (3150 GPU days), making it more suitable for practical applications. Through the use of EA for architecture search, MFGENAS effectively navigates high-dimensional parameter spaces, maximizing its computational resource usage. In conclusion, MFGENAS exhibits superior performance and considerable advantages in parameter efficiency and search costs.

In the comparative algorithms presented in Table 3, MOEA-PS [76] is a study published in 2023. MOEA-PS utilizes an adjacency list to represent the internal structure of neural networks. Additionally, an innovative mechanism is incorpo-rated within the multi-objective genetic algo-rithm to guide the crossover and mutation pro-cesses. Furthermore, neural networks are gener-ated using a surrogate model stacking structural block, which employs a limited search space. A queue technique also eliminates the worst-performing network structures, significantly re-ducing the number of architectures that require training. However, despite the change in how architectures are represented, the method still trains the architectures for full epochs, resulting in a requirement of 2.6 GPU days on CIFAR-10. In contrast, the proposed MFGENAS leverages multi-fidelity evaluation combined with a genetic decision controller to selectively filter offspring, thereby accelerating the evaluation process while ensuring the quality of the generated architec-tures. The results obtained from experiments on the CIFAR-100 dataset further confirm the ex-ceptional performance of MFGENAS, as detailed in Table 4. MFGENAS recorded a test error of 16.42%, the best among all compared algo-rithms, and significantly lower than NASNet-

A (16.58%) and E-NAS (17.27%), demonstrating its effectiveness with complex datasets. Furthermore, MFGENAS has a parameter count of 2.97M, which is lower than other high-performing models such as AmoebaNet-B (3.2M) and NAO (10.8M), indicating greater efficiency in resource usage. Importantly, the search cost for MFGENAS is merely 0.3 GPU days, a figure that is considerably lower than those of AmoebaNet-B (3150 GPU days) and Large-Scale Evolution (2750 GPU days), illustrating its superiority in real-world applications. The strategy of utilizing EA for architecture search allows MFGENAS to main-tain high classification accuracy while signifi-cantly lowering the demand for computational resources. Overall, MFGENAS demonstrates not only remarkable performance on the CIFAR-100 dataset but also provides a novel method for deep neural network search with minimized cost and parameter requirements. As shown in Table 4, GENAS [75] is a study published in 2025 that utilizes gradient-guided evolution to search for neural network architec-tures. GENAS extracts topological structures from a supernet by combining global and local search operators. It is specifically designed to overcome the limitations of traditional evolution-ary and gradient-based NAS methods. Thanks to its algorithmic framework, GENAS enables the performance evaluation of candidate architec-tures without the need for retraining, thereby significantly reducing the computational cost of NAS. This characteristic allows GENAS to achieve a fast search speed—requiring only 0.26 GPU days on CIFAR-100—while reaching a test error rate of 16.86%. However, the method relies on the construction of a supernet prior to the search. Pre-trained supernets provided by other researchers can be reused for standard image classification tasks.

In contrast, building a domain-specific supernet can be time-consuming and labor-intensive when applying this method to other domains. To address this issue, we propose MFGENAS, which integrates multi-fidelity evaluation with a genetic decision controller to accelerate the evaluation process. Unlike GENAS, MFGENAS does not depend on pre-trained supernets and can efficiently discover high-quality neural architectures with low time consumption across a wide range of application domains.

4.3 Visualization of Found Architectures and Iterative Process

Figure 7 presents the objective space comprised of the fitness values from six populations throughout the MFGENAS search conducted on CIFAR-10. We focused on iterations that best illustrate the search dynamics: the initial objective space, the first iteration, the eighth iteration, the thirteenth iteration, the eighteenth iteration, and the final iteration.

At initialization, the points are dispersed throughout the space, with two in the bottom right corner and a wide distribution on the left, highlighting the success of our initialization strategy in generating a variety of individual architectures. By the first iteration, the points start to converge toward the lower left corner, and the outlying points on the right have vanished. This indicates the effectiveness of our search strategy in retaining better solutions while discarding fewer effective ones.

Throughout the eighth, thirteenth, and eighteenth iterations, the points continue to move closer to the lower left corner. This indicates that our search strategy converges and does not fall into local optima.

By the final iteration, the blue points fully occupy the Pareto front, exhibiting superior performance indicators compared to the other points.

Additionally, the distribution of the blue points is more uniform, achieving enhanced dispersion along the Pareto front.

4.4 Architecture Scalability Analysis

To illustrate the search capabilities of MFGENAS, we applied the architectures found by MFGENAS on CIFAR-10 to the ImageNet dataset [], assessing the scalability of the identified neural architectures. Table 5 summarizes the scalability evaluation of MFGENAS on the ImageNet dataset in comparison with other leading models. MFGENAS recorded a Top-1 error rate of 25.96% and a Top-5 error rate of 8.06%, demonstrating strong performance across multiple algorithms, especially regarding parameter utilization and search cost.

In contrast to NASNet-A and NASNet-B, which have error rates of 26.00% and 27.2% respectively, MFGENAS achieves a lower Top-1 er-

Table 3. Performance evaluation of MFGENAS compared to other similar models on the CIFAR-10 dataset.

Algorithm	**Test Error (%)**	**Parameters (M)**	**Search Cost (GPU days)**	**Search Method**
Wide ResNet [51]	4.17	36.5	–	Manual
DenseNet-BC [8]	3.46	25.6	–	Manual
EAS [56]	4.23	23.4	10	RL
Block-QNN-S [54]	4.38	6.1	96	RL
NASNet-A [10]	2.65	3.3	2000	RL
MetaQNN [28]	6.92	11.2	80	RL
E-NAS [55]	2.89	4.6	0.5	RL+weight sharing
DARTS (second order) [57]	2.76	3.3	4	Gradient based
DARTS (first order) [57]	3.00	3.3	1.5	Gradient based
NAO [58]	3.18	10.6	200	Gradient based
SQNAS [60]	2.76	3.10	0.06	Gradient based
VNAS [61]	2.43	3.78	0.3	Gradient based
AER [62]	2.53	3.74	0.3	Gradient based
NAO+WS [58]	3.53	2.5	0.3	Gradient based+weight sharing
DDPNAS [78]	2.44	–	1.8	Gradient based
Large-Scale Evolution [65]	5.40	5.4	2750	Evolution
AmoebaNet-A [63]	3.34	3.2	3150	Evolution
NSGA-Net [68]	2.75	3.3	4	Evolution
Hierarchical Evolution [64]	3.63	15.7	300	Evolution
Genetic-CNN [67]	7.10	–	17	Evolution
AE-CNN [70]	4.30	2.0	27	Evolution
CNN-GA [66]	4.78	2.9	35	Evolution
SI-ENAS [72]	4.07	–	1.8	Evolution
E2EPP [69]	5.30	–	8.5	Evolution
GENAS [75]	2.45	3.53	0.26	Evolution
CARS-E [73]	2.86	3.0	0.4	Evolution+weight sharing
MOEA-PS [76]	2.77	3.0	2.6	Evolution
MFGENAS	**2.39±0.09**	**2.94**	**0.3**	**Evolution**

Table 4. Performance evaluation of MFGENAS compared to other similar models on the CIFAR-100 dataset.

Algorithm	**Test Error (%)**	**Parameters (M)**	**Search Cost (GPU days)**	**Search Method**
Wide ResNet [51]	20.50	36.5	–	Manual
DenseNet-BC [8]	17.18	25.6	–	Manual
NASNet-A [10]	16.58	3.3	2000	RL
MetaQNN [28]	27.14	11.2	80	RL
Block-QNN-S [54]	20.65	6.1	96	RL
E-NAS [55]	17.27	4.6	0.5	RL + weight sharing
NAO [58]	15.67	10.8	200	Gradient based
Large-Scale Evolution [65]	23.00	40.4	2750	Evolution
AmoebaNet-B [63]	15.80	3.2	3150	Evolution
NSGA-Net [68]	20.74	3.3	8	Evolution
Genetic-CNN [67]	29.03	–	17	Evolution
AE-CNN [70]	20.85	5.4	36	Evolution
CNN-GA [66]	20.53	4.1	40	Evolution
SI-ENAS [72]	18.64	–	1.8	Evolution
E2EPP [69]	22.02	–	8.5	Evolution
GENAS [75]	16.86	3.53	0.26	Evolution
MOEA-PS [76]	18.97	5.8	5.2	Evolution
MFGENAS	**16.42±0.37**	**2.97**	**0.3**	**Evolution**

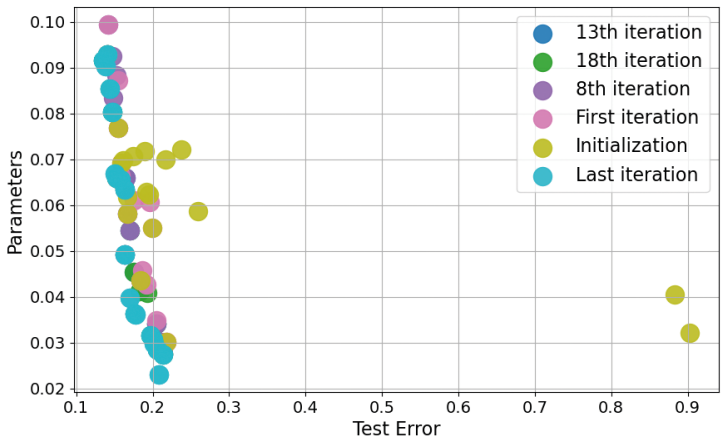

Figure 7. The objective space composed of the population fitness values (parameters (MB), test error) from six iterations of MFGENAS on CIFAR-10.

ror rate with a parameter count of 5.98M, which is significantly less than NAO (11.4M) and the AmoebaNet models (5.1M for AmoebaNet-A and 5.3M for AmoebaNet-B). This highlights the parameter efficiency of MFGENAS. Additionally, the search cost for MFGENAS stands at just 0.3 GPU days, considerably less than the extensive resource requirements of NASNet (2000 GPU days) and AmoebaNet (3150 GPU days). This finding underscores MFGENAS 's ability to deliver high-performance models with reduced computational demands, thereby improving its potential for application in large datasets. Overall, MFGENAS not only demonstrates remarkable classification performance on the ImageNet dataset but also provides a scalable solution through efficient parameter usage and minimal search costs.

4.5 Ablation Analysis

This section will utilize ablation analysis to validate the effectiveness of the proposed multi-fidelity evaluation based on a genetic decision controller. The variant of MFGENAS that does not incorporate this evaluation will be designated as MFGENAS (base). We will compare the performance of architectures obtained from MFGENAS (base) with those from the full MFGENAS model on the CIFAR-10 and CIFAR-100 datasets.

Table 6 summarizes the results from the ablation experiments conducted on the CIFAR-10 and CIFAR-100 datasets. This experiment contrasts MFGENAS (base), which does not implement genetic decision controller-based multi-fidelity evaluation, with the complete MFGENAS model. In the CIFAR-10 dataset, MFGENAS (base) records a test error of 2.30%, with 2.87M parameters and a search cost of 2.55 GPU days. Meanwhile, the complete MFGENAS model achieves a test error of 2.39% with a slight increase in parameter count to 2.94M, but significantly reduces the search cost to only 0.3 GPU days. While the complete model has a marginally higher test error on CIFAR-10, its computational efficiency is clearly enhanced.

In the CIFAR-100 dataset, MFGENAS (base) presents a test error of 16.18%, with 2.89M parameters and a search cost of 2.55 GPU days. In contrast, the complete MFGENAS model shows a test error of 16.42% and a slightly increased parameter count of 2.97M, while still maintaining a search cost of 0.3 GPU days. These results indicate that although the complete model experiences a slight increase in test error, the significant gains in search cost and parameter efficiency validate the effectiveness of the multi-fidelity evaluation mechanism in practical applications.

4.6 Strategy Validation of Genetic Decision Controller

In this research, we employ a KELM as the genetic decision controller due to its excellent interpretability, allowing for a clear visualization of feature significance and decision-making. In the context of NAS, various architectural features, such as layer count, node count, and activation functions, can be effectively analyzed by the KELM, which captures the relationships between these features and accuracy through simple rules. This enables the effective identification of the best architecture. To highlight the contribution of the KELM in this study, we will conduct comparisons with other comparable machine learning approaches.

The training dataset is comprised of multiple distinct individual architectures. The performance comparison results of two individual architectures are used as labels. For example, A single data encoding consists of a pair of architectures; if the performance of the architecture encoded earlier surpasses that of the latter, a label of 1 is assigned to the encoding. Conversely, if the latter architecture performs better, a label of 2 is assigned.

During the NAS search process, evaluation is conducted by pairing two architectures together, and a KELM model is employed to predict the label, thereby determining the relative superiority of the two architectures. This configuration allows the KELM model to perform exceptionally well for several reasons. Primarily, KELM has a robust capacity to model nonlinear relationships, which leads to high prediction accuracy.

Additionally, KELM demonstrates good performance on small sample sizes. Given that acquiring individual neural architecture data is time-intensive—requiring the full training of each architecture to derive accurate labels—we limit our training dataset for the KELM to just 260 trained architectures. In comparison to other models, such as support vector machines and multi-layer percep-

Table 5. Architecture scalability performance evaluation of MFGENAS compared to other similar mod-els on the ImageNet dataset.

Algorithm	Top-1 Error (%)	Top-5 Error (%)	Parameters (M)	Search Cost (GPU days)	Search Method
Inception-v1 [52]	30.20	10.10	6.6	-	Manual
ShuffleNet (v1) [53]	29.10	10.20	5	-	Manual
NASNet-B [10]	27.2	8.7	5.3	2000	RL
NASNet-A [10]	26.00	8.4	5.3	2000	RL
NASNet-C [10]	27.5	9	4.9	2000	RL
NAO [58]	25.70	8.20	11.4	200	Gradient based
DARTS[57]	26.70	8.70	4.7	4	Gradient based
AmoebaNet-B [63]	26.00	8.50	5.3	3150	Evolution
AmoebaNet-A [63]	25.50	8.00	5.1	3150	Evolution
AmoebaNet-C [63]	24.30	7.60	6.4	3150	Evolution
CARS-E [73]	26.30	8.40	4.4	0.4	Evolution
Genetic-CNN [67]	27.87	9.74	156.0	17	Evolution
MFGENAS	25.96±1.26	8.06±0.58	5.98	0.3	Evolution

Table 6. Ablation experiment. (MFGENAS (base) refers to the version of MFGENAS that does not utilize multi-fidelity evaluation based on the genetic decision controller, while MFGENAS represents the complete model.)

Datasets	Algorithm	Test Error (%)	Parameters (M)	Search Cost
CIFAR-10	MFGENAS (base)	2.30	2.87	2.55
	MFGENAS	**2.39**	**2.94**	**0.3**
CIFAR-100	MFGENAS (base)	16.18	2.89	2.55
	MFGENAS	**16.42**	**2.97**	**0.3**

tron, KELM is better at mitigating overfitting in scenarios with limited data, thereby improving generalization. Moreover, KELM offers a relatively simple and efficient training and prediction process, which is particularly beneficial in high-dimensional feature spaces like those involved in NAS. This efficiency allows for quick iterations and the effective identification of optimal architectures. In conclusion, the KELM's superior interpretability, strong nonlinear modeling capabilities, small sample proficiency, and computational efficiency make it the most suitable choice.

We also employ an improved swarm intelligence algorithm, CPLO, to optimize the kernel parameters of KELM, further enhancing its predictive capability. Table 7 and Fig. 8 presents the performance comparison between CPLO-KELM and several other models (K-nearest neighbor (KNN), categorical regression tree (CART), BP, Support Vector Machine (SVM), Light Gradient Boosting Machine (LightGBM) and PLO-KELM).

In this section's diagnostic experiments, the parameter settings for both GORUN-KELM and RUN-KELM are kept consistent to ensure a fair comparison. Specifically, the maximum number of iterations is set to 10, and the population size is fixed at 50. The regularization parameter C and the kernel parameter γ are chosen from the range $[2^(-5), 2^5]$. Among the classification methods applied, the KNN is utilized with K=1, and distance is calculated using the Euclidean metric. The CART, an enhanced decision tree based on the ID3 algorithm, is employed with MATLAB's default parameter settings. The BP neural network, a classical learning algorithm, is also implemented using the Levenberg–Marquardt optimization technique. The number of hidden neurons for the BP model is set to 8, and training stops when the mean squared error reaches 0.001. Prior to model training, all data are normalized to the range [-1, 1]. A ten-fold cross-validation strategy is adopted to assess model performance. LightGBM employed the gradient boosting decision tree as the boosting type, which builds an ensemble of decision trees iteratively to minimize the loss function. The number of boosting rounds was fixed at 100; however, to prevent overfitting, an early stopping strategy was implemented with patience of 10 rounds, halting training if no improvement was observed on the validation set during this interval. These parameter choices balance training efficiency and model generalization, allowing the model to learn effectively from the training data while mitigating overfitting risks.

The evaluation of the models is based on four standard metrics: Accuracy (ACC), Sensitivity, Specificity, and the Matthews Correlation Coefficient (MCC), which are formally defined as follows:

$$ACC = \frac{TP+TN}{TP+FP+FN+TN} \tag{13}$$

$$Sensitivity = \frac{TP}{TP+FN} \tag{14}$$

$$Specificity = \frac{TN}{FP+TN} \tag{15}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{16}$$

ACC primarily measures the overall correctness of the classification model. Sensitivity, also known as the true positive rate, indicates the model's ability to identify positive cases, thereby minimizing missed diagnoses correctly. Specificity, on the other hand, reflects the model's capability to correctly recognize negative cases, effectively reducing false positives or misdiagnosed. It is worth noting that an inherent trade-off exists between sensitivity and specificity: enhancing one often leads to a compromise in the other. In other words, efforts to lower the rate of missed diagnoses may inadvertently raise the rate of misdiagnoses and vice versa.

MCC serves as a comprehensive indicator of model robustness, offering a balanced evaluation even in cases of class imbalance.

Table 7 comprehensively compares classification performance across seven models, including KNN, CART, BP, SVM, LightGBM, PLO-KELM and CPLO-KELM. The evaluation metrics used are ACC, Sensitivity, Specificity, and MCC, averaged over 10-fold cross-validation. In terms of overall classification accuracy, CPLO-KELM achieved the best performance with a mean ACC of 0.8790 and the lowest standard deviation (0.0176), demonstrating both high accuracy and robustness. KNN and SVM followed with mean accuracies of 0.8620

Table 7. The results obtained by CPLO-KELM and other models.

Model	Indicator	Mean	Std	1	2	3	4	5	6	7	8	9	10
KNN		0.8620	0.0387	0.8600	0.8200	0.9000	0.8600	0.8200	0.8100	0.9200	0.9200	0.8400	0.8700
CART		0.7940	0.0398	0.8600	0.7400	0.8000	0.7500	0.7400	0.7900	0.7900	0.8500	0.8100	0.8100
BP		0.8470	0.0490	0.8100	0.8700	0.8600	0.8100	0.8600	0.7900	0.9100	0.9400	0.7800	0.8400
SVM	ACC	0.8550	0.1248	0.9100	0.8700	0.8700	0.8900	0.8700	0.8300	0.9400	0.9800	0.8900	0.5000
LightGBM		0.8486	0.0455	0.9064	0.9034	0.8084	0.8906	0.8097	0.8152	0.9152	0.8119	0.8097	0.8155
PLO-KELM		0.8310	0.0624	0.8300	0.8000	0.8700	0.8200	0.6700	0.8400	0.8800	0.9100	0.8200	0.8700
CPLO-KELM		0.8790	0.0176	0.8500	0.8800	0.8700	0.8900	0.8500	0.9100	0.8800	0.8800	0.8900	0.8900
KNN		0.7426	0.0869	0.9231	0.6364	0.7333	0.6667	0.7857	0.6667	0.8000	0.8333	0.7143	0.6667
CART		0.7123	0.1085	0.8462	0.6364	0.8667	0.8333	0.7143	0.6667	0.5000	0.6667	0.6429	0.7500
BP		0.7530	0.1404	0.9231	0.6364	0.6000	0.8333	0.8571	0.6111	0.7000	1.0000	0.7857	0.5833
SVM	Sensitivity	0.5572	0.2159	0.6923	0.3636	0.3333	0.5000	0.5714	0.2778	0.5000	0.8333	0.5000	1.0000
LightGBM		0.7685	0.1206	0.8351	0.8458	0.5417	0.8356	0.5274	0.8408	0.8392	0.8379	0.7404	0.8408
PLO-KELM		0.8452	0.0798	0.9231	0.9091	0.8000	0.9167	0.9286	0.8889	0.8000	0.8333	0.7857	0.6667
CPLO-KELM		0.8554	0.0608	0.8462	0.9091	0.8000	0.8333	0.7857	0.8889	0.8000	1.0000	0.8571	0.8333
KNN		0.8793	0.0385	0.8506	0.8427	0.9294	0.8864	0.8256	0.8415	0.9333	0.9255	0.8605	0.8977
CART		0.8042	0.0438	0.8621	0.7528	0.7882	0.7386	0.7442	0.8171	0.8222	0.8617	0.8372	0.8182
BP		0.8618	0.0546	0.7931	0.8989	0.9059	0.8068	0.8605	0.8293	0.9333	0.9362	0.7791	0.8750
SVM	Specificity	0.9116	0.0565	0.9425	0.8326	0.8647	0.8432	0.9186	0.8512	0.9889	0.9894	0.9535	0.9318
LightGBM		0.8786	0.0066	0.8723	0.8734	0.8673	0.8800	0.8872	0.8845	0.8854	0.8797	0.8717	0.8844
PLO-KELM		0.8276	0.0784	0.8161	0.7865	0.8824	0.8068	0.6279	0.8293	0.8889	0.9149	0.8256	0.8977
CPLO-KELM		0.8336	0.0404	0.8506	0.7865	0.8824	0.8523	0.7791	0.7927	0.8889	0.8723	0.7907	0.8409
KNN		0.5159	0.0819	0.6009	0.3680	0.6300	0.4678	0.4836	0.4508	0.6340	0.5546	0.4814	0.4883
CART		0.3888	0.0950	0.5660	0.2684	0.5056	0.3953	0.3410	0.4186	0.2373	0.3341	0.3958	0.4264
BP		0.4970	0.0887	0.5256	0.4568	0.4809	0.4686	0.5830	0.3907	0.5650	0.6842	0.4276	0.3877
SVM	MCC	0.4793	0.1609	0.6153	0.3089	0.3923	0.4603	0.4762	0.3074	0.6176	0.8227	0.5029	0.2891
LightGBM		0.5613	0.0460	0.6115	0.6251	0.6149	0.5192	0.5165	0.5291	0.5283	0.5216	0.6174	0.5291
PLO-KELM		0.5173	0.0585	0.5536	0.4797	0.5882	0.5236	0.3881	0.6021	0.5379	0.5284	0.4836	0.4883
CPLO-KELM		0.5811	0.0458	0.5486	0.6797	0.5882	0.5294	0.6276	0.5569	0.5379	0.5392	0.5905	0.6130

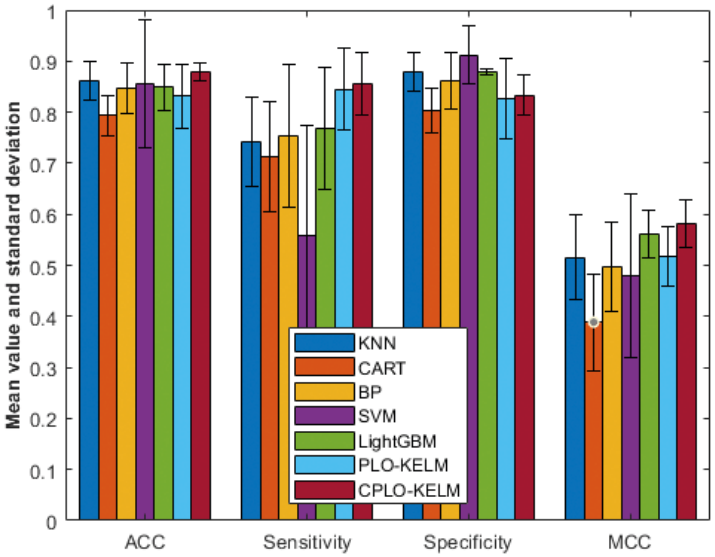

Figure 8. Performance verification experiment of CPLO-KELM on architectural data.

and 0.8550, respectively. Notably, SVM exhibited a relatively large variance (std = 0.1248), indicating less stable performance across folds. Regarding Sensitivity, which reflects the model's ability to correctly identify positive samples, CPLO-KELM (0.8554) and PLO-KELM (0.8452) outperformed other methods. This highlights the effectiveness of the proposed optimization strategies in reducing missed diagnoses. In contrast, SVM showed the lowest Sensitivity (0.5572), suggesting it tends to misclassify positive instances, which is further corroborated by its MCC value (0.4793). Specificity, indicating the ability to correctly identify negative samples, was highest for SVM (0.9116), followed closely by LightGBM (0.8786) and KNN (0.8793). This reveals that SVM is conservative in predicting positives, which may explain its high Specificity but low Sensitivity. Finally, in terms of MCC, which captures the overall balance and reliability of binary classification, CPLO-KELM achieved the highest value (0.5811), indicating a well-balanced prediction across both classes. LightGBM and PLO-KELM also showed competitive MCC values of 0.5613 and 0.5173, respectively.

Although CPLO-KELM achieves superior performance in terms of ACC, sensitivity, and MCC, it exhibits a relatively lower specificity compared to some other classifiers such as SVM and LightGBM. This phenomenon can be attributed to the model's tendency to favor positive class predictions, which enhances its sensitivity. Specially, reducing the number of false negatives, but may come at the cost of increasing false positives, thereby lowering specificity. From a technical standpoint, CPLO-KELM integrates a greedy selection mechanism and crossover mechanism, both of which are designed to enhance the model's exploration of the decision boundary, particularly focusing on hard-to-classify or minority class samples. In datasets with class imbalance or subtle inter-class variation, such mechanisms may bias the classifier toward over-predicting the minority or positive class to ensure that as few positives as possible are missed. While this behavior improves sensitivity and overall detection capability, it also increases the likelihood of misclassifying negative instances as positive, which directly impacts specificity. Moreover, since specificity reflects the true negative rate, any misclassification of negative samples—especially if the negative class is dominant—can disproportionately reduce this metric. Thus, the trade-off observed between sensitivity and specificity is a common issue in medical diagnosis and imbalanced classification tasks, where prioritizing the reduction of missed diagnoses (false negatives) is often more critical than minimizing false alarms (false positives).

4.7 Strategy Validation of CPLO

To enhance the predictive capabilities of the KELM in this study, we utilized an advanced swarm intelligence approach for optimizing its hyperparameters, which we refer to as CPLO. To evaluate the optimization effectiveness of CPLO, we compared it against other swarm intelligence techniques on the IEEE

CEC 2017 benchmark. IEEE CEC 2017 is a commonly used standard for comparing swarm intelligence algorithms, which allows for a clear demonstration of CPLO's performance advantages.

Table 8 details the performance results of CPLO alongside several algorithms, including PLO, PSO, DE, GWO, and WOA. The table includes "rank" to denote the ranking of each algorithm and "AVG" for the average ranking after 30 trials. The symbols +, =, and - indicate the comparative outcomes of CPLO against the other algorithms across 30 CEC 2017 test func-tions: "+" indicates CPLO outperformed the comparison algorithm, "-" indicates it underper-formed, and "=" indicates no significant perfor-mance difference.

Table 8 provides a comparison of CPLO's performance against other optimization algorithms on the IEEE CEC 2017 benchmark functions, detailing rankings, relative performance indicators, and average scores. CPLO achieved the highest ranking (Rank 1) with an average score of 1.67. The "+/=/-" symbols indicate CPLO's relative performance across test functions: for example, CPLO outperformed the original PLO on 14 functions, showed no significant difference on 5 functions, and underperformed on 11 functions. PLO ranked second with an average score of 1.93. This result demonstrates the effectiveness of the cross-mechanism introduced to improve PLO. Compared to other well-known swarm intelligence methods, CPLO shows significant advantages. For instance, it outperforms PSO on 26 test functions, DE on 22 test functions, and both GWO and WOA across all test func-

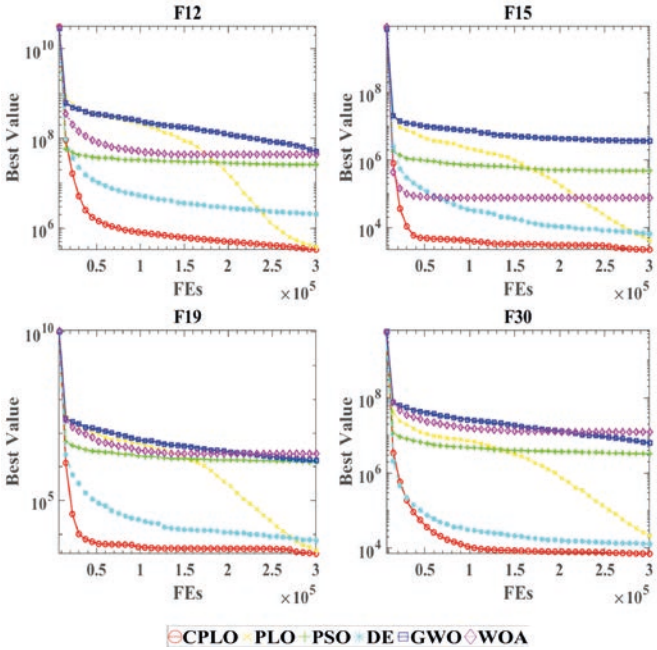

Figure 9. The convergence curves of CPLO and other comparable optimization algorithms on the IEEE CEC 2017 benchmark.

tions. These results highlight CPLO's strong competitive performance, especially against GWO and WOA, and its consistent convergence characteristics across a variety of test functions.

Table 8. The ranking of CPLO and other comparable optimization algorithms on the IEEE CEC 2017 benchmark.

Algorithm	Rank	+/=/-	AVG
CPLO	**1**		**1.67**
PLO	2	14/5/11	1.93
PSO	4	26/2/2	4.27
DE	3	22/2/6	2.97
GWO	5	30/0/0	4.63
WOA	6	30/0/0	5.53

In Figure 9, we present the convergence curves of CPLO alongside five other optimization algorithms for four test functions from IEEE CEC 2017. The horizontal axis indicates the iteration count, and the vertical axis displays the optimized fitness values. The functions under consideration are F12, F15, F19, and F30. The red curve depicts the performance of CPLO, clearly showing that it converges to lower fitness values compared to the other algorithms, reflecting its enhanced effectiveness. Furthermore, CPLO achieves a quicker convergence, reaching the bottom of the graph sooner than its counterparts.

Conclusion

In this work, we propose an ENAS method (MFGENAS), which accelerates evaluation by multi-fidelity evaluation and a genetic decision controller. We established initialization and individual encoding that enhance the diversity of the population. MFGENAS uses the NASGA-II framework with accuracy and architectural parameters as multi-optimization targets. Through a genetic decision controller and multi-fidelity evaluation approach, MFGENAS can efficiently search superior deep neural network architectures while consuming minimal time. A genetic decision controller is employed to assess the quality of offspring generated by MFGENAS to reduce the number of expensive evaluations. If the offspring is to outperform the parent, an expensive evaluation is then performed. We adopt KELM optimized by CPLO to select the genetic decision controller. This approach dramatically accelerates the ENAS search process. We validated the performance of MFGENAS through rigorous experiments, including comparisons with peer-to-peer competitors on CIFAR-10 and CIFAR-

100, achieving a test error of 2.39 on CIFAR-10 with only 0.3 GPU days of computation. Moreover, we assessed the scalability of architectures on ImageNet, visualized the MFGENAS search process, and conducted an ablation analysis of the genetic decision controller approach.

In future research, we will strive to develop more efficient ENAS. There is potential to enhance the prediction accuracy of the genetic decision controller utilized in this study. Additionally, the design of the search space remains crucial for ENAS. By investigating various search space designs, we aim to expand the application of ENAS into other fields, such as medicine and cognitive diagnosis.

Declarations

Acknowledgments: This work was supported by National Natural Science Foundation of China (62476030), the“Pioneering Leadership + X” Research and Development Plan of Zhejiang Provincial Department of Science and Technology (2024C03237) and the Natural Science Foundation of Hangzhou (2024SZRYBH180010).

Authors contribution statement: Zenglin Qiao: Writing - Original Draft, Writing - Review & Editing, Software, Validation, Visualization, Investigation, Formal Analysis; Ali Asghar Heidari: Formal Analysis, Investigation, Software, Writing - Review & Editing, Validation, Visualization; Xinchao Zhao: Writing - Review & Editing, Visualization, Funding Acquisition, Software, Project Administration; Huiling Chen: Writing - Review & Editing, Funding Acquisition, Software, Resources.

Data availability and access: The datasets generated during and analyzed during the current study are available from the corresponding author on reasonable request.

Ethics and Consent to Participate declarations: Not applicable.

Conflict of interest: There are no conflicts of interest to declare.

Declaration of AI and AI-assisted technologies in the writing process During the preparation of this work, the author(s) used ChatGPT to improve and proofread the English of the paper. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

References

[1] N. H. Luong, Q. M. Phan, A. Vo, T. N. Pham, and D. T. Bui, ”Lightweight multiobjective evolutionary neural architecture search with low-cost proxy metrics,” Information Sciences, vol. 655, p. 119856, 2024/01/01/ 2024, doi: https://doi.org/10.1016/j.ins.2023.119856.

[2] X. Ma, W. Tang, P. Wang, X. Guo, and L. Gao, ”Extracting stage-specific and dynamic modules through analyzing multiple networks associated with cancer progression,” IEEE/ACM transactions on computational biology and bioinformatics, vol. 15, no. 2, pp. 647-658, 2016.

[3] J. Huang et al., ”Deep reinforcement learning-based trajectory pricing on ride-hailing platforms,” ACM Transactions on Intelligent Systems and Technology (TIST), vol. 13, no. 3, pp. 1-19, 2022.

[4] Z. Song et al., ”Robustness-aware 3d object detection in autonomous driving: A review and outlook,” IEEE Transactions on Intelligent Transportation Systems, 2024.

[5] C. Hema and F. P. G. Marquez, ”Emotional speech recognition using cnn and deep learning techniques,” Applied Acoustics, vol. 211, p. 109492, 2023.

[6] V. Uc-Cetina, N. Navarro-Guerrero, A. Martin-Gonzalez, C. Weber, and S. Wermter, ”Survey on reinforcement learning for language processing,” Artificial Intelligence Review, vol. 56, no. 2, pp. 1543-1575, 2023.

[7] K. He, X. Zhang, S. Ren, and J. Sun, ”Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778, 2016.

[8] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, ”Densely connected convolutional networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700-4708, 2017.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, ”Rethinking the inception architecture for computer vision,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818-2826, 2016.

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, ”Learning transferable architectures for scalable image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697-8710, 2018.

[11] Y. Chen et al., "Renas: Reinforced evolutionary neural architecture search," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 4787-4796, 2019.

[12] B. Lyu et al., "Efficient multi-objective neural architecture search framework via policy gradient algorithm," Information Sciences, vol. 661, p. 120186, 2024/03/01/ 2024, doi: https://doi.org/10.1016/j.ins.2024.120186.

[13] Y. Shen et al., "Proxybo: Accelerating neural architecture search via bayesian optimization with zero-cost proxies," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, no. 8, pp. 9792-9801, 2023.

[14] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 1946-1956, 2019.

[15] N. Sinha and K.-W. Chen, "Neural Architecture Search Using Covariance Matrix Adaptation Evolution Strategy," Evolutionary Computation, vol. 32, no. 2, pp. 177-204, 2024, doi: 10.1162/evco_a_00331.

[16] Y. Li, R. Liu, X. Hao, R. Shang, P. Zhao, and L. Jiao, "EQNAS: Evolutionary Quantum Neural Architecture Search for Image Classification," Neural Networks, vol. 168, pp. 471-483, 2023/11/01/ 2023, doi: https://doi.org/10.1016/j.neunet.2023.09.040.

[17] S. Xiao, B. Zhao, and D. Liu, "Semi-supervised accuracy predictor-based multi-objective neural architecture search," Neurocomputing, vol. 609, p. 128472, 2024/12/07/ 2024, doi: https://doi.org/10.1016/j.neucom.2024.128472.

[18] Q. M. Phan and N. H. Luong, "Enhancing multi-objective evolutionary neural architecture search with training-free Pareto local search," Applied Intelligence, vol. 53, no. 8, pp. 8654-8672, 2023/04/01 2023, doi: 10.1007/s10489-022-04032-y.

[19] L. Xu, J. Zheng, C. He, J. Wang, B. Zheng, and J. Lv, "Adaptive Multi-particle Swarm Neural Architecture Search for High-incidence Cancer Prediction," IEEE Transactions on Artificial Intelligence, 2025.

[20] L. Wen, L. Gao, X. Li, and H. Li, "A new genetic algorithm based evolutionary neural architecture search for image classification," Swarm and Evolutionary Computation, vol. 75, p. 101191, 2022.

[21] V. Lopes, M. Santos, B. Degardin, and L. A. Alexandre, "Guided evolutionary neural architecture search with efficient performance estimation," Neurocomputing, vol. 584, p. 127509, 2024/06/01/ 2024, doi: https://doi.org/10.1016/j.neucom.2024.127509.

[22] J. Liu, R. Cheng, and Y. Jin, "Bi-fidelity evolutionary multiobjective search for adversarially robust deep neural architectures," Neurocomputing, vol. 550, p. 126465, 2023/09/14/ 2023, doi: https://doi.org/10.1016/j.neucom.2023.126465.

[23] S. Wang, Z. Liu, J. Li, M. Gong, and R. Yang, "Evolutionary Multitasking Collaborative Neural Architecture Search for Scene Classification," in 2024 IEEE Congress on Evolutionary Computation (CEC), 30 June-5 July 2024 2024, pp. 1-8, doi: 10.1109/CEC60901.2024.10612042.

[24] Y. Gao et al., "HGNAS++: Efficient Architecture Search for Heterogeneous Graph Neural Networks," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 9, pp. 9448-9461, 2023, doi: 10.1109/TKDE.2023.3239842.

[25] H. Wang, C. Ge, H. Chen, and X. Sun, "PreNAS: Preferred One-Shot Learning Towards Efficient Neural Architecture Search," presented at the Proceedings of the 40th International Conference on Machine Learning, Proceedings of Machine Learning Research, [Online]. Available: https://proceedings.mlr.press/v202/wang23f.html, 2023.

[26] L. Ma, H. Kang, G. Yu, Q. Li, and Q. He, "Single-Domain Generalized Predictor for Neural Architecture Search System," IEEE Transactions on Computers, vol. 73, no. 5, pp. 1400-1413, 2024, doi: 10.1109/TC.2024.3365949.

[27] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," Journal of artificial intelligence research, vol. 4, pp. 237-285, 1996.

[28] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning," in International Conference on Learning Representations, 2022.

[29] R. Negrinho and G. Gordon, "Deeparchitect: Automatically designing and training deep architectures," arXiv preprint arXiv:1704.08792, 2017.

[30] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in European conference on machine learning, Springer, pp. 282-293, 2006.

[31] B. Lyu, S. Wen, K. Shi, and T. Huang, "Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing," IEEE

Transactions on Cybernetics, vol. 53, no. 2, pp. 1158-1169, 2021.

[32] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," arXiv preprint arXiv:1812.09926, 2018.

[33] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," arXiv preprint arXiv:1812.00332, 2018.

[34] Y. Xu et al., "Pc-darts: Partial channel connections for memory-efficient architecture search," arXiv preprint arXiv:1907.05737, 2019.

[35] R. S. Sukthanker, A. Krishnakumar, M. Safari, and F. Hutter, "Weight-entanglement meets gradient-based neural architecture search," arXiv preprint arXiv:2312.10440, 2023.

[36] L. Xie et al., "ZO-DARTS++: An Efficient and Size-Variable Zeroth-Order Neural Architecture Search Algorithm," arXiv preprint arXiv:2503.06092, 2025.

[37] Y. Wang et al., "MedNAS: Multiscale Training-Free Neural Architecture Search for Medical Image Analysis," IEEE Transactions on Evolutionary Computation, vol. 28, no. 3, pp. 668-681, 2024, doi: 10.1109/TEVC.2024.3352641.

[38] Z. Fan, J. Wei, G. Zhu, J. Mo, and W. Li, "Evolutionary neural architecture search for retinal vessel segmentation," arXiv preprint arXiv:2001.06678, 2020.

[39] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," IEEE transactions on neural networks and learning systems, vol. 34, no. 2, pp. 550-570, 2021.

[40] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," Evolutionary computation, vol. 28, no. 1, pp. 141-163, 2020.

[41] S. Jiang, Z. Ji, G. Zhu, C. Yuan, and Y. Huang, "Operation-level early stopping for robustifying differentiable NAS," Advances in Neural Information Processing Systems, vol. 36, pp. 70983-71007, 2023.

[42] K. Sakamoto, H. Ishibashi, R. Sato, S. Shirakawa, Y. Akimoto, and H. Hino, "Atnas: Automatic termination for neural architecture search," Neural Networks, vol. 166, pp. 446-458, 2023.

[43] I. Trofimov, N. Klyuchnikov, M. Salnikov, A. Filippov, and E. Burnaev, "Multi-Fidelity Neural Architecture Search With Knowledge Distillation," IEEE Access, vol. 11, pp. 59217-59225, 2023, doi: 10.1109/ACCESS.2023.3234810.

[44] J. Sun, W. Yao, T. Jiang, and X. Chen, "Efficient search of comprehensively robust neural architectures via multi-fidelity evaluation," Pattern Recognition, vol. 146, p. 110038, 2024/02/01/ 2024, doi: https://doi.org/10.1016/j.patcog.2023.110038.

[45] J. Liu, J. Yan, H. Xu, Z. Wang, J. Huang, and Y. Xu, "Finch: Enhancing Federated Learning With Hierarchical Neural Architecture Search," IEEE Transactions on Mobile Computing, vol. 23, no. 5, pp. 6012-6026, 2024, doi: 10.1109/TMC.2023.3315451.

[46] W. Wang, X. Zhang, H. Cui, H. Yin, and Y. Zhang, "FP-DARTS: Fast parallel differentiable neural architecture search for image classification," Pattern Recognition, vol. 136, p. 109193, 2023/04/01/ 2023, doi: https://doi.org/10.1016/j.patcog.2022.109193.

[47] Z. Chen, G. Qiu, P. Li, L. Zhu, X. Yang, and B. Sheng, "MNGNAS: Distilling Adaptive Combination of Multiple Searched Networks for One-Shot Neural Architecture Search," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 11, pp. 13489-13508, 2023, doi: 10.1109/TPAMI.2023.3293885.

[48] T. Zhang et al., "NASRec: Weight Sharing Neural Architecture Search for Recommender Systems," presented at the Proceedings of the ACM Web Conference 2023, Austin, TX, USA, 2023. [Online]. Available: https://doi.org/10.1145/3543507.3583446.

[49] C. Peng, Y. Li, R. Shang, and L. Jiao, "RSBNet: One-shot neural architecture search for a backbone network in remote sensing image recognition," Neurocomputing, vol. 537, pp. 110-127, 2023/06/07/ 2023, doi: https://doi.org/10.1016/j.neucom.2023.03.046.

[50] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[51] S. Zagoruyko, "Wide residual networks," arXiv preprint arXiv:1605.07146, 2016.

[52] C. Szegedy et al., "Going deeper with convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9, 2015.

[53] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 6848-6856, 2018.

[54] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture

generation," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2423-2432, 2018.

[55] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in International conference on machine learning, PMLR, pp. 4095-4104, 2018.

[56] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in Proceedings of the AAAI conference on artificial intelligence, vol. 32, no. 1, 2018.

[57] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," arXiv preprint arXiv:1806.09055, 2018.

[58] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," Advances in neural information processing systems, vol. 31, 2018.

[59] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "Dpp-net: Device-aware progressive search for pareto-optimal neural architectures," in Proceedings of the European conference on computer vision (ECCV), pp. 517-531, 2018.

[60] G. Biju and G. Pillai, "Sequential node search for faster neural architecture search," Knowledge-Based Systems, vol. 300, p. 112145, 2024.

[61] B. Ma, J. Zhang, Y. Xia, and D. Tao, "VNAS: variational neural architecture search," International Journal of Computer Vision, vol. 132, no. 9, pp. 3689-3713, 2024.

[62] K. Jing, L. Chen, and J. Xu, "An architecture entropy regularizer for differentiable neural architecture search," Neural Networks, vol. 158, pp. 111-120, 2023.

[63] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in Proceedings of the aaai conference on artificial intelligence, vol. 33, no. 01, pp. 4780-4789, 2019.

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

[65] E. Real et al., "Large-scale evolution of image classifiers," in International conference on machine learning, PMLR, pp. 2902-2911, 2017.

[66] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," IEEE transactions on cybernetics, vol. 50, no. 9, pp. 3840-3854, 2020.

[67] L. Xie and A. Yuille, "Genetic cnn," in Proceedings of the IEEE international conference on computer vision, pp. 1379-1388, 2017.

[68] Z. Lu et al., "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, pp. 4750-4754, 2021.

[69] 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 Transactions on Evolutionary Computation, vol. 24, no. 2, 2020.

[70] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely Automated CNN Architecture Design Based on Blocks," IEEE transactions on neural networks and learning systems, vol. 31, no. 4, pp. 1242-1254, 2020.

[71] C. He, H. Tan, S. Huang, and R. Cheng, "Efficient evolutionary neural architecture search by modular inheritable crossover," Swarm and Evolutionary Computation, vol. 64, p. 100894, 2021.

[72] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Sampled training and node inheritance for fast evolutionary neural architecture search," arXiv preprint arXiv:2003.11613, 2020.

[73] Z. Yang et al., "Cars: Continuous evolution for efficient neural architecture search," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1829-1838, 2020.

[74] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," arXiv preprint arXiv:1804.09081, 2018.

[75] Y. Xue, X. Han, F. Neri, J. Qin, and D. Pelusi, "A gradient-guided evolutionary neural architecture search," IEEE transactions on neural networks and learning systems, 2024.

[76] Y. Xue, C. Chen, and A. Słowik, "Neural architecture search based on a multi-objective evolutionary algorithm with probability stack," IEEE Transactions on Evolutionary Computation, vol. 27, no. 4, pp. 778-786, 2023.

[77] Y. Tian, S. Peng, S. Yang, X. Zhang, K. C. Tan, and Y. Jin, "Action Command Encoding for Surrogate-Assisted Neural Architecture Search," IEEE Transactions on Cognitive and Developmental Systems, vol. 14, no. 3, 2022.

[78] X. Zheng et al., "Ddpnas: Efficient neural architecture search via dynamic distribution pruning," International Journal of Computer Vision, vol. 131, no. 5, pp. 1234-1249, 2023.

Zenglin Qiao received the Master's degree from the College of Disaster Prevention Science and Technology. Currently, he is a current PhD student at Beijing University of Posts and Telecom-munications. Her research interests include NAS and artificial intelligence.
https://orcid.org/0000-0002-3793-7150

Ali Asgher Heidari received the Ph.D. degree in geospatial infor-mation systems from the University of Tehran, Tehran, Iran, in 2020. He has been an exceptionally talented researcher with the School of the Computing National University of Singapore, Uni-versity of Tehran, Wenzhou university, and Iran's National Elites Foundation. His research interests include advanced machine learning, evolutionary computation, performance optimization, and information systems. For more information, researchers can refer to his website https://aliasgharheidari.com.
https://orcid.org/0000-0001-6938-9948

Xinchao Zhao received the Ph.D. degree from the Key Laboratory of Mathematics Mechanization, Chinese Academy of Sciences, Beijing, China, in 2005. He is currently a Professor with the School of Science, Beijing University of Posts and Telecommunications. He has published more than 100 research papers in journals and conferences. His research interests include evolutionary computa-tion, swarm intelligence, and engineering applications.
https://orcid.org/0000-0001-9376-7646

Huiling Chen is currently a professor in the college of computer science and artificial intelligence at Wenzhou University, China. He received his Ph.D. degree in the department of computer sci-ence and technology at Jilin University, China. His present research interests center on evolutionary computation, machine learning, data mining, and their applications to medical diagnosis, bankruptcy prediction, and parameter extraction of the solar cell. With more than 40000 citations and an H-index of 101, he is ranked worldwide among top scientists for Computer Science & Electronics prepared by Guide2Research, the best portal for com-puter science research (https://guide2research.com/u/huiling-chen). He was served as the co-editor-in-chief of Computers in Biology and Medicine (2021.10-2024.12), and the editorial board member of Scientific Reports. He has published more than 200 papers in international journals and conference proceedings, including IEEE Transactions on Circuits and Systems for Video Technology, IEEE System Journal, IEEE Internet of Things Journal, Future Genera-tion Computer Systems, Pattern Recognition, Expert Systems with Applications, Knowledge-Based Systems, and others. He has more than 40 ESI highly cited papers and 10 hot cited papers.
https://orcid.org/0000-0002-7714-9693