BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 04/14 pp45–64
DOI: 10.26599/BDMA.2024.9020036
Volume 8, Number 1, February 2025

GPT-NAS: Neural Architecture Search Meets Generative Pre-Trained Transformer Model

Caiyang Yu, Xianggen Liu, Yifan Wang, Yun Liu, Wentao Feng, Xiong Deng, Chenwei Tang*, and Jiancheng Lv*

Abstract: The pursuit of optimal neural network architectures is foundational to the progression of Neural Architecture Search (NAS). However, the existing NAS methods suffer from the following problem using traditional search strategies, i.e., when facing a large and complex search space, it is difficult to mine more effective architectures within a reasonable time, resulting in inferior search results. This research introduces the Generative Pre-trained Transformer NAS (GPT-NAS), an innovative approach designed to overcome the limitations which are inherent in traditional NAS strategies. This approach improves search efficiency and obtains better architectures by integrating GPT model into the search process. Specifically, we design a reconstruction strategy that utilizes the trained GPT to reorganize the architectures obtained from the search. In addition, to equip the GPT model with the design capabilities of neural architecture, we propose the use of the GPT model for training on a neural architecture dataset. For each architecture, the structural information of its previous layers is utilized to predict the next layer of structure, iteratively traversing the entire architecture. In this way, the GPT model can efficiently learn the key features required for neural architectures. Extensive experimental validation shows that our GPT-NAS approach beats both manually constructed neural architectures and automatically generated architectures by NAS. In addition, we validate the superiority of introducing the GPT model in several ways, and find that the accuracy of the neural architecture on the image dataset obtained from the search after introducing the GPT model is improved by up to about 9%.

Key words: Neural Architecture Search (NAS); Generative Pre-trained Transformer (GPT) model; evolutionary algorithm; image classification

1 Introduction

In recent years, Deep Neural Networks (DNNs) have shown impressive fitting power in various tasks, ranging across computer vision[1], natural language processing[2], etc. In addition, they have been widely used in different industries, such as medicine[3–7] and industry[8]. More recently, the emergence of large models[9, 10] has provided an extremely effective path to solving all kinds of tasks. However, the training of

• Caiyang Yu, Xianggen Liu, Yifan Wang, Yun Liu, Wentao Feng, Chenwei Tang, and Jiancheng Lv are with College of Computer Science, and Engineering Research Center of Machine Learning and Industry Intelligence (affiliated to the Ministry of Education), Sichuan University, Chengdu 610065, China. E-mail: yucy@scu.edu.cn; liuxianggen@scu.edu.cn; wangyifan5217@scu.edu.cn; yliu@scu.edu.cn; Wtfeng2021@scu.edu.cn; tangchenwei@scu.edu.cn; lvjiancheng@scu.edu.cn.
• Xiong Deng is with Department of Mechanical Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA. E-mail: dxiong@stevens.edu.
* To whom correspondence should be addressed.
Manuscript received: 2024-02-29; revised: 2024-04-18; accepted: 2024-05-20

large models is extremely resource intensive, and in addition, large models are not suitable for use on some lightweight devices. Therefore, some small models are still extremely necessary. In Deep Learning (DL), it is widely accepted that neural networks with individual architectures present different inductive biases. Although multiple advanced architectures have been designed, the intrinsic principles of the architectural building remain unclear. As a result, researchers usually spend a large time manually seeking the neural architectures that are suitable to the given tasks.

To accelerate the designing process and improve the quality of the neural architectures, Neural Architecture Search (NAS)[11] has emerged as one of the effective methods to design the optimal neural network architecture automatically. The main advantage of NAS lies in its ability to automate the tedious and time-consuming process of designing neural architectures. Additionally, NAS can improve the quality of neural architectures by leveraging search strategies to find architectures that achieve better performance than human-designed architectures.

Currently, based on different optimization techniques, the mainstream NAS search strategies include Reinforcement Learning (RL)[12, 13], Evolutionary Algorithm (EA)[14], and Gradient Optimization (GO)[15]. Algorithms, such as NAS-RL[12], MetaQNN[13], and Block-QNN-S[16], all belong to the first category. For different RL methods, the key lies in how to design the agent's policy and the corresponding optimization process[17]. For example, Zoph and Le[12] used the RNN policy to select the basic information and form the architecture, while the proximal policy is used for optimization in subsequent work[18]. Secondly, the EA-based NAS (EA-NAS) searches for the optimal architecture mainly by the properties of the algorithm. For example, in Ref. [19], the Genetic Algorithm (GA) is used as the optimization strategy to complete the algorithm search process, while in Ref. [20], the genetic programming strategy is adopted. Finally, GO-based NAS is a category of algorithms that do not rely on any strategy. It mainly implements search in a continuous search space[15].

However, with the rapid growth of the NAS space, the limitations of traditional search strategies are becoming increasingly apparent, especially in the context of an expanding search space. Firstly the explosive growth in the number of architectures increases the difficulty of searching with traditional strategies, e.g., the diversity of network structures and connections can easily construct millions of architectures in the search space, which poses a greater challenge to the efficiency of traditional strategies. Secondly, traditional search strategies are carried out in isolation, and the efficiency of the search only relies on the optimization ability of the strategy itself, which has a great limitation in the search perspective, and it is difficult to consider the search problem from multiple perspectives or a global perspective.

As a result, several improvement efforts have been generated to optimize the search strategy. For example, in Ref. [21], Zhang et al. proposed an adaptive scalable NAS method based on the reinforced I-Ching divination evolutionary algorithm and a variable-architecture encoding strategy. It simplifies the reinforcement learning algorithm and enhances the search efficiency of evolutionary algorithms, addressing the nonconvexity problem in NAS. In addition, Maziarz et al.[22] proposed evolutionary-neural hybrid agents to blend deep reinforcement learning with evolutionary algorithms for improved NAS. This method outperforms both neural and evolutionary agents in terms of accuracy and search cost. However, the method described above combining different search strategies suffers from the following problems.

(1) Complexity and overfitting. The integration of multiple strategies often results in increased complexity, which can lead to models that are highly specialized to the training data, risking overfitting and reduced generalizability.

(2) Integration challenges. Effectively combining different strategies requires careful balance and tuning. There's a risk of one strategy overpowering another, leading to suboptimal outcomes.

To this end, we propose a novel approach, called GPT-NAS, by introducing the Generative Pre-trained Transformer (GPT) model in NAS to improve search efficiency. This simplifies the process and eliminates the complexity associated with integrating multiple policies. Additionally, it can compensate for the limitations of existing strategies instead of competing with them. Without disturbing the search process of the original search strategy, we use the GPT model to reconstruct each architecture obtained from the search. Specifically, the layer structures in the architecture are

randomly masked and the GPT model is utilized to regenerate the masked layers based on the contextual information in the architecture. However, the use of regular GPT models suffers from two challenges: (1) the inability to directly recognize neural architectures; (2) the lack of specialized neural architecture design capabilities. To address the above challenges, we first introduce an encoding strategy to map the neural architecture into a textual representation to facilitate the recognition of the GPT model. For the latter, the GPT model is allowed to be trained on the neural architecture datasets, so that it can effectively learn techniques and principles for designing superior neural architectures.

We evaluate our model on the CIFAR-10, CIFAR-100[23], and ImageNet-1K[24]. Experimental results show that GPT-NAS can demonstrate excellent accuracy on three datasets. Further analysis shows that after introducing the GPT model, the architecture accuracy obtained through search can be improved by up to 9%, and the search efficiency can be significantly improved. This proves that the introduction of the GPT model compensates for the insufficient search performance of traditional search strategies. The contributions of this article are presented as follows.

(1) For the first time, we propose a new NAS algorithm, called GPT-NAS, which utilizes the GPT model to optimize the neural architecture obtained from the search, compensating for the inadequate search efficiency of the search strategy.

(2) GPT-NAS allows GPT models to be trained on a neural architecture dataset to learn the principles of superior neural architecture design, enabling the reconstruction of neural architectures.

(3) Extensive experiments have proven that GPT-NAS demonstrates state-of-the-art experimental results on three datasets. Furthermore, we demonstrate that the introduced GPT model improves the efficiency of traditional search.

2 Related Work

In this section, we aim to describe two key areas relevant to this study, i.e., NAS and GPT models.

2.1 NAS

NAS is a subfield of machine learning that focuses on automating the design of neural network architectures. It involves three key components: a search space that defines the potential architectures, a search strategy to explore this space, and a performance estimation strategy to evaluate the architectures. NAS aims to discover optimal network structures for specific tasks, reducing the need for manual design and expertise. In Section 1, we have introduced the search strategy, and below we will discuss the search space in detail.

The search space in NAS is a critical aspect that defines the range of architectures that can be explored. It typically includes layer-based, block-based, and cell-based search spaces, each catering to different levels of architectural granularity. Layer-based search space focuses on the individual layers of a neural network and the type, size, and connectivity are variables in the search. This search space is the simplest and is one of the most commonly used methods in the early days of NAS[12]. Block-based search space is defined in terms of blocks, which are larger structures than individual layers. Each block can contain multiple layers with predefined connections. For example, Lu et al.[25] utilized a conformer-based search space for multi-task audio separation, where the search involves varying the number of blocks, heads, and channels to optimize the architecture. Cell-based search space is a more granular approach, where the architecture in search space consists of cells, which are small network, sub-structures. Cells are repeated to form the final architecture. Pouy et al.[26] proposed a cell-based hierarchical search space, aiming to optimize search time and handle a wide range of state-of-the-art Convolutional Neural Network (CNN) architectures. Another study by Jin et al.[27] introduces a dual attention mechanism in the cell-based search space, enhancing the interrelationships between layers within the architecture. Each of these search spaces offers different levels of flexibility and complexity, influencing the efficiency and effectiveness of the NAS process.

2.2 GPT

The advent of GPT has revolutionized various fields by providing advanced natural language processing capabilities. The applications of GPT models are diverse and continually expanding, as they offer significant improvements in understanding, generating, and interacting with human language. In the medical field, GPT-4 has exceeded expectations in competency examinations, indicating its potential in medical education and practice[28]. In genomics, GPT models, like GeneTuring, have been employed to reduce AI

hallucinations, improving accuracy in genomic data analysis[29]. Furthermore, in data science and education, GPT models facilitate model selection and personalized learning, separately, transforming these fields with AI-driven solutions[30]. In addition, the application of GPT to neural networks has been extensively studied. One notable example is the GPT-GNN framework, which leverages GPT for the pre-training of Graph Neural Networks (GNNs), demonstrating significant improvements in modelling graph-structured data[31]. Zheng et al.[32] explored the ability of GPT-4 to conduct NAS through an approach named GENIUS, which uses GPT-4's generative capabilities to navigate the architecture search space efficiently. Additionally, the integration of GPT-4 in Graph Neural Architecture Search (GNAS) showcases the model's capability to automate and refine the process of designing graph neural networks, leading to more accurate and efficient architectures[33]. However, all of the above approaches use the inherent capabilities of GPT to optimize the architecture, and do not make the GPT model train on the neural architecture dataset, while learning and understanding the architecture. This study addresses this issue and proposes to train the GPT model using neural architecture as a dataset to assist NAS.

3 Proposed Method

In this section, we describe our proposed method in detail. First, the overview of the algorithm is given in Section 3.1. Then, in the subsequent sections, we will describe the encoding approach of the neural architecture (Section 3.2), the pre-training and fine-tuning methods of the GPT model (Section 3.3), and an introduction to the overall NAS framework (Section 3.4).

3.1 Algorithm overview

To compensate for the limitations of traditional search strategies in the face of a large and complex search space, we propose the framework GPT-NAS, which uses the GPT model to optimize the NAS algorithm. It is widely accepted that the pre-trained GPT model is extremely gifted at predicting text, and this study takes advantage of it. By training GPT models on neural architecture datasets, the goal is to learn the principles of neural architecture design and transfer them to specific tasks. Specifically, we divide the content of the framework into three procedures, i.e., neural architecture encoding, pre-training and fine-tuning the GPT model, and neural architecture search.

Neural architecture encoding. In order for the GPT model to recognize neural architectures, it is necessary to encode them. The encoding strategy translates the neural architecture into the form of characters, where each character corresponds to a specific operation in the architecture, such as a convolutional layer, a fully connected layer, etc.

Pre-Training and fine-tuning the GPT model. Pre-training and fine-tuning are two critical procedures for developing high-performance GPT models. Let the GPT model be pre-trained on a neural architecture dataset to equip it with the design capabilities of superior neural architectures and fine-tuned on specific tasks.

Neural architecture search. The neural architecture search procedure consists of two parts, namely architecture search and reconstruction. For the former, the architectures are sampled, trained, and evaluated using GA as the search strategy. For the latter, the sampled architectures are reorganized using GPT. **Note that GA is used as the search strategy in this paper, but other search strategies can also be used, such as RL, EA, etc. While the GPT model is used to assist the search process, it does not affect the original search strategy.**

The core of the GPT-NAS framework, consisting of the three producers described above, lies in the optimization of the architecture obtained from the search using the GPT model. Through continuous iteration, the optimal architecture is found. In addition, during the training of the neural architecture, we propose a strategy to accelerate the training process and reduce the training time.

3.2 Neural architecture encoding

Effective encoding of neural architectures facilitates fast recognition of architectures by GPT models. As a result, designing a general encoding strategy to accommodate the CNN architecture is necessary.

Inspired by Ref. [34], we divide the structures that make up the CNN architecture into four categories: convolutional layer, pooling layer, Fully Connected (FC) layer, and other layers. Among them, the first three structures are necessary for almost all CNN architectures, and the last one is used to represent all the remaining structures not included in the first three, such as the activation function.

(1) Convolutional layer: The convolutional layer is the most fundamental and vital structure in CNNs. In convolutional operations, the core technology is the use of convolutional kernels (filters), which aim to extract features from the input image. The convolution kernel is a two-dimensional matrix (height and width) and the parameters can be learned. In addition, the convolution kernel slides in the horizontal and vertical directions of the original image according to the "stride". In general, regardless of the value of "stride" (when the size of the convolution kernel is not 1), the newly obtained feature map will certainly be smaller than the size of the original image, and this is also not conducive to the edge information of the image to work (because the convolution kernel will compute the central region of the image several times, while the edge region is relatively less). Therefore, the surrounding "padding" of the image is needed to solve the above problem. Commonly, the convolution kernel convolves on each channel of the input feature map, which is a channel-dense connection. In contrast, there is a channel sparse connection, which groups the input feature map channels and convolves each group separately. This process is called "groups" and has the advantage of effectively reducing the number of parameters. In most cases, the convolutional kernel size needs to be enlarged if the respective field of a larger feature map is desired. However, the consequent drawback is that the number of parameters increases, so "dilation" appears, which is an operation that injects space into the standard convolution kernel. In conclusion, the properties of the convolution layer are input size, output size, convolution kernel size, stride size, the number and value of padding, the space size of the kernel, the number of groups for the channels in the input feature map, and whether to use bias term.

(2) Pooling layer: The properties of the pooling layer are very similar to those of the convolutional layer, except that the following details need to be changed. First, the pooling layer contains two types, i.e., max pooling (MAX) and average pooling (AVG), so the property "type" needs to be introduced to indicate which type is chosen. Second, the purpose of pooling is to reduce the size of the feature map, but this process has no parameters to learn, so there is no need for property "groups" to reduce the number of parameters. Third, as we all know, the purpose of "padding" is to expand the values in all four directions of the feature map, so not only the values but also the quantities need to be defined. However, in the pooling layer, the value of "padding" is not set manually but the default value of 0. If not, it will affect the selection of feature values and make the final result biased. In summary, compared with the convolutional layer, the property "groups" is removed and a new property "type" is added, so the pooling layer still maintains ten properties.

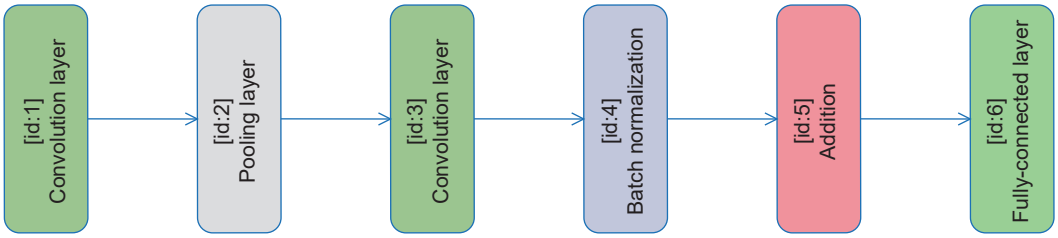
(3) Fully-connected layer: Compared to the above two structures of the network, the fully layer is straightforward to express. The fully-connected layer has only two properties, i.e., "in_size" and "out_size". In many neural architectures for vision-related tasks, a fully layer is necessary, such as in image classification tasks, where the final result is output for classification only through a fully-connected layer. Therefore, the structure is simple but essential.

(4) Other layers: In the CNN architecture, there are many structures with different functions, such as activation function, Batch Normalization (BN) , etc. These structures play the role of catalysts in the CNN architecture and enhance the performance of the neural architecture. Therefore, in this part, all relevant structures will be described. The common properties of these structures are the "name", "in_size", "out_size", and "value". Among them, "value" denotes the relevant parameter involved in the corresponding structure.

As shown above, we have described the properties of different structures in CNN. In our methodology, encoding the CNN architecture into a textual format is a crucial step that facilitates the integration of neural networks with the GPT model. The coding process converts the architectural specifications of each layer into a structured string format, allowing for efficient processing by the GPT model. As detailed in Table 1, the unique properties of each layer, such as identifier, type, and dimensionality, are systematically transformed into a delimited text string that encapsulates the layer's configuration. For example, the first layer of network architecture in Fig. 1, a convolutional layer, is encoded as follows: "id:1;in_size:32-32-3;out_size:32-32-3;kernel:1-1;stride:1-1;padding:0-0-0-0;dilation:1;groups:1;bias_used:No". This encoding string begins with a unique identifier (id:1), followed by input and output sizes (in_size and out_size), kernel size (kernel), and other relevant parameters such as stride, padding, dilation,

Table 1 Properties of different layers in CNN.

No.	Name	Conv	Pooling	FC	Others	Remark
1	id	✓	✓	✓	✓	An identifier with an integer value
2	type	–	✓	–	–	A string value
3	name	–	–	–	✓	A string value
4	in_size	✓	✓	✓	✓	Input size of a three-element integer tuple
5	out_size	✓	✓	✓	✓	Output size of a three-element integer tuple
6	kernel	✓	✓	–	–	A two-element integer tuple
7	stride	✓	✓	–	–	A two-element integer tuple
8	padding	✓	✓	–	–	A four-element integer tuple
9	dilation	✓	✓	–	–	An integer
10	groups	✓	–	–	–	An integer
11	value	–	–	–	✓	A tuple
12	bias_used	✓	✓	–	–	A boolean number




```
id:1;in_size:32-32-3;out_size:32-32-3;kernel:1-1;stride:1-1;padding:0-0-0-0;dilation:1;groups:1;bias used:No
id:2;type:Max;in_size:32-32-3;out_size:16-16-3;kernel:2-2;stride:2-2;padding:1-0-1-0;dilation:1;bias used:No
id:3;in_size:16-16-3;out_size:8-8-64;kernel:2-2;stride:2-2;padding:0-0-0-0;dilation:1;groups:1;bias used:No
id:4;name:BN;in_size:16-16-3;out_size:16-16-3;value:Null
id:5;name:Addition;in_size:16-16-3;out_size:16-16-3;value:Null
id:6;in_size:768;out_size:10
```

Fig. 1 Description of the CNN architecture.

groups, and whether a bias is used. By employing semicolons to delimit properties and hyphens to specify dimensions or parameter values, we create a standardized format that the GPT model can readily interpret and use for further processing tasks such as architecture reconstruction and optimization. In addition, the encoding for a max pooling layer is similarly structured but includes its specific parameters, such as the pooling type and the relevant kernel and stride sizes, reflecting its purpose and design within the overall architecture. The encoded string 'id:2;type:Max;in_size:32-32-3; out_size:16-16-3; kernel:2-2; stride:2-2; padding:1-0-1-0; dilation:1; bias_used:No' details a max pooling layer designed to reduce the spatial dimensions of the preceding convolutional layer's output. By precisely encoding each layer, we can capture the intricate details that define the structure and behaviour of a CNN. This meticulous process not only standardizes the representation of neural architectures, but also paves the way for the use of GPT models to analyze, optimize, and ultimately generate neural architecture designs that are both innovative and effective for a variety of tasks.

3.3 Pre-training and fine-tuning the GPT model

Providing the ability to design neural architectures for GPT models is at the heart of this research. Therefore, pre-training and fine-tuning the GPT model using the neural architecture as training data can effectively achieve what is needed in this paper.

The pre-training phase of a GPT model requires training on a large amount of data to learn the parameter distribution, followed by a fine-tuning phase to suit various tasks. As a result, the mainstream approach will use unsupervised learning for maximum likelihood estimation in the first phase and use supervised learning to optimize the model using a

cross-entropy loss function in the second phase. However, in this study, since both the pre-training and fine-tuning phases are trained on the neural architectural dataset and ground truth is presented, it will be a better choice to use a supervised learning approach for both phases. The specific implementation process is shown in Fig. 2.

In line with the previously mentioned, we leverage the GPT model to predict the next data by the previous information, i.e., the structural information of the previous layers to predict the structural information of the next layer (see Section 3.4 for details). Thus, based on the given layer structures, we will minimize the following objective function:

$$\begin{cases} \mathcal{F} = \mathcal{L}\left(\hat{L}_t, L_t \middle| L_{<t}, C\right), \\ C = f\ (\theta, \mathcal{S}) \end{cases} \tag{1}$$

where C denotes the neural architectures, θ is the parameter used in constituting the neural architecture, $\mathcal{S}$ is the corresponding network layer structures, and $f(\ )$ denotes the combination strategy. In the objective function $\mathcal{F}$, $\mathcal{L}$ denotes the loss function, $\hat{L}_t$ and L_t denote the predicted layer structure and the true layer structure obtained at layer t, respectively, and $L_{<t} = (L_{t-k}, L_{t-k-1}, \ldots, L_{t-1})$ (k is the size of the data window). The loss function $\mathcal{L}$ is defined as the cross-entropy loss function,

$$\mathcal{L} = -\frac{1}{T-k} \sum_{t=k+1}^{T} \sum_{i=1}^{N} L_t \log\left(\hat{L}_t[i]\right) \tag{2}$$

where T denotes the total number of layers in C, N denotes the number of categories in the layer structure, and $\hat{L}_t[i]$ denotes the probability of the i-th category in the predicted layer structure $\hat{L}_t$.

3.4 NAS

Based on the obtained GPT model, in this section, we will describe in detail how to effectively combine the GPT model with NAS to improve the optimization efficiency. The procedure of neural architecture search consists of two main parts, i.e., network architecture search and architecture reconstruction, and Fig. 3 shows the flowchart.

(1) NAS: In the first part, we implement the search of neural architectures with the GA as the search strategy. Specifically, multiple individuals are initialized randomly, and each individual is represented as a neural architecture. Then, each neural architecture is reconstructed and the performance of the reconstructed architecture is evaluated. After that, during the iterative process, the GA is used to perform evolutionary operations on individuals, including crossover, mutation, and selection strategies to facilitate the acquisition of better-performance individuals and form a new population. In more concrete terms, for the crossover operation, two individuals are randomly selected from the population as parents, and the offspring is obtained by exchanging the information of the two individuals. For the mutation operation, the information of the individual is randomly changed to obtain a new individual (both operations are detailed in Ref. [35]). **Note that the new architectures obtained after the evolution operation in each iteration still need to be reconstructed and evaluated.** Finally, the optimal individual will be obtained after reaching the

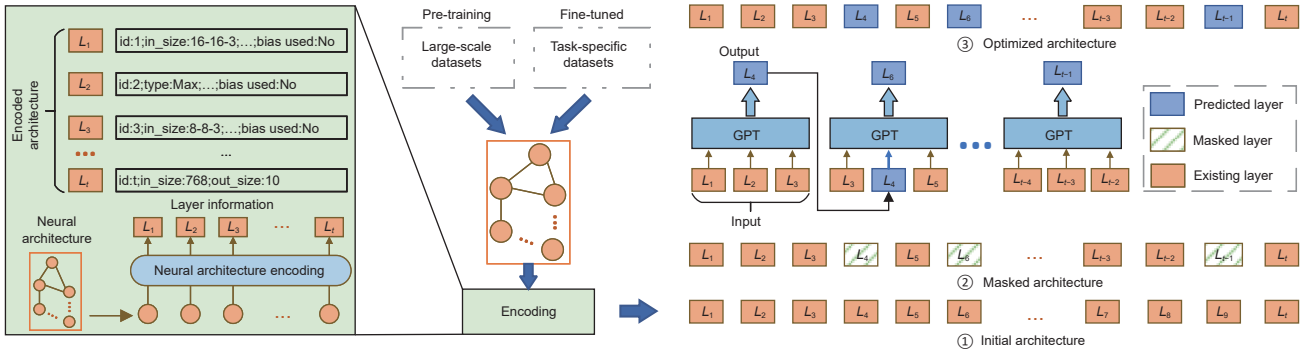


Fig. 2 Pre-training and fine-tuning process for the GPT models. The pre-training and fine-tuning phases are the same process, with the difference being that the neural architectures are obtained from different datasets. The graph depicts how the network architecture is integrated with the GPT model during the pre-training and fine-tuning phases. Specifically, for the masked network architectures, *k* layers are input, and then the next layer is predicted by the GPT model, and this operation is repeated until all masked layers are filled.

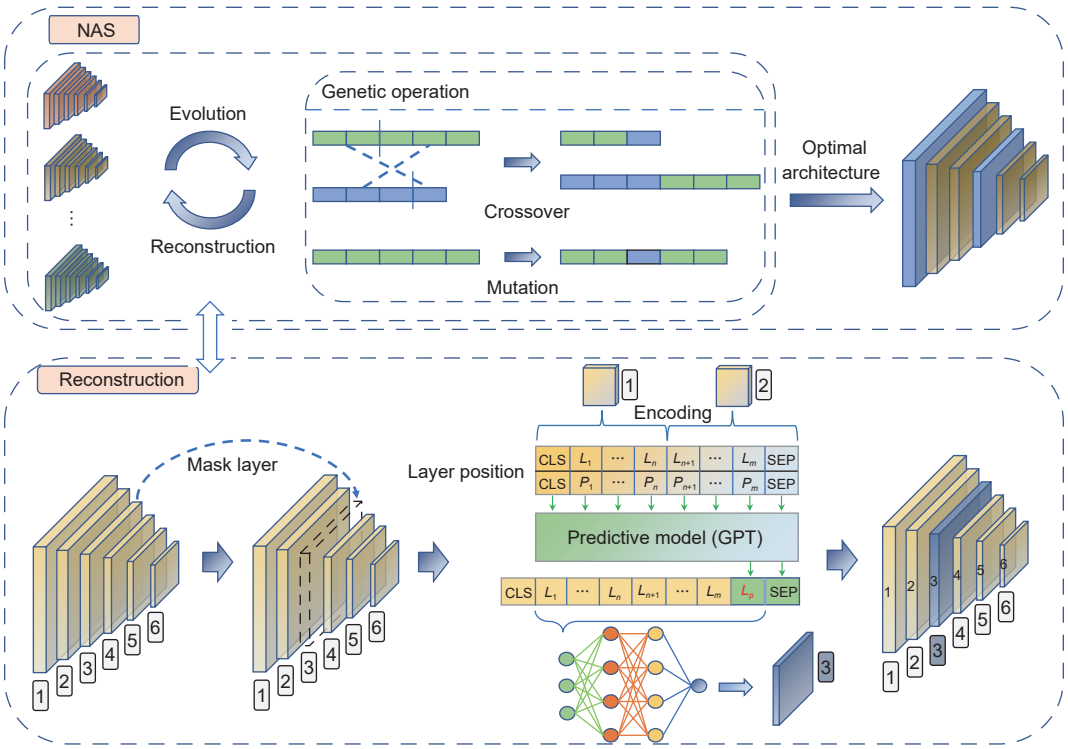


Fig. 3 Flowchart of neural architecture search. We divide this procedure into two parts, i.e., search and reconstruction. The former is a generalized neural architecture search method using GA as the search strategy. The latter is a reorganization of the structure of the neural architecture by the GPT model (where P_i (i=1, 2, ..., m) denotes the layer position).

maximum number of iterations.

(2) Architecture reconstruction: In the second part, we will describe how the GPT model can be used to optimize the architecture obtained from the search. Note that to reduce the cost and complexity of the reconstruction, we will build the neural architecture based on blocks rather than layers. Specifically, we start choosing one or more layer structures. Then, the GPT model comes into play and predicts a new layer structure to replace the masked layer using the pre-existing structure information as a reference. In the following, we use a neural architecture as an example to illustrate the process of reconstruction. As shown at the bottom half of Fig. 3, randomly selecting a layer for masking (assuming the layer is in the third block), and then the fine-tuned GPT model is used to predict the layer structure based on the layers in first and second blocks (including layer and location information). However, since the composition of the neural architecture is based on blocks and the predictions obtained by the GPT model are the layer, we introduce a Fully Connected Network (FCN) to select the fittest blocks according to the layers and replace the third block. Specifically, a new layer obtained through the GPT model is integrated with the previous layers. This combined set of layers is input into the FCN to obtain the corresponding block structure. Note that we treat the process as a classification process, treating different blocks as different categories. The details can be seen in the Algorithm 1.

In summary, the neural architecture obtained from each iteration is optimized using the GPT model, and performance improvement is achieved by changing its structure. In this case, we propose the concept of mask rate to determine, whether a layer of the neural architecture is selected and whether the corresponding block is masked. However, according to the law of GA, the quality of the offspring population will be better than that of the parent population. Therefore, as the iteration proceeds, the mask rate of the structures will be linearly decreasing. The mask rate at the t-th iteration is as follows:

$$\text{rate}_t = \text{rate}_{\text{ori}} - \frac{\text{iter}_t}{\text{iter}_{\max}} \times \text{rate}_{\text{ori}} \tag{3}$$

where rate_{ori} indicates the initialized mask rate, iter_t and $\text{iter}_{\max}$ denote the t-th iteration and the number of iterations, respectively.

Algorithm 1 Structure prediction

Input: Neural architecture with masked structures cnn_mask
Output: Neural architecture with optimization cnn_new

```
for (1 ⩽ L ⩽ len (cnn_mask)) do
    if layer is masked then
        data ← transform L_i into textual data (i∈[index–k, index); //index denotes serial number of lays
        new_layer ← prediction layer using GPT based on data;
        new_block ← prediction block using FCN based on data and new layer;
        cnn_new ← cnn_new∪new_block;
    else
        cnn_new ← cnn_new∪L;
    end
end
return cnn_new
```

3.5 Acceleration strategy

One of the criticisms of NAS development has been the time-consuming problem. In a recent study, Ref. [36] proposes that the performance of neural architectures can be evaluated without training. However, the method suffers from assumptions and does not guarantee that the final results are the same as the real ones. Therefore, an adequate search for each architecture remains the dominant approach. In this study, we introduce a new acceleration strategy to reduce the time cost of the GPT-NAS, as shown below.

Only the structures obtained from the prediction are trained. Training the entire neural architecture is time-consuming, but it is more efficient if only the reconstructed structures of the architecture are trained. To speed up the training while ensuring that the performance of the neural architecture is not lost, we assume that the layer structures obtained from the GPT model predictions are 'vital' structures and propose to train only these structures (confirmed in Section 5.2.2). And for neural architectures without predicted structures, inspired by Ref. [37], we will train BN in the neural architecture. Finally, if both of the above rules are not satisfied, the overall neural architecture is trained.

Only a small number of epochs are trained. This strategy has been covered in some works[18, 38]. While in this study, a smaller number of epochs will be used. The rationale for doing so is motivated by warmup[39], which makes the learning rate increase in fewer epochs to alleviate the model overfitting phenomenon and reach an equilibrium state. If the model stabilizes faster (i.e., the higher accuracy rate achieved) during this time, it can be considered a better performance of the model (confirmed in Section 5.2.2).

4 Experimental Design

To verify the effectiveness of the proposed algorithm, we conduct a series of experiments. Therefore, this section will present the design of all the elements involved in the experiments. First, we introduce the competitive algorithm for comparison with GPT-NAS (Section 4.1). Then, the datasets used in this experiment and the hyper-parameters are introduced (Section 4.2). Finally, we describe the parameter settings in the experiment (Section 4.3).

4.1 Peer competitors

To verify the effectiveness of the proposed algorithm, we selected several competitive algorithms for comparison in our experiments. We divide the selected peer competitors into two categories, i.e., algorithms obtained by manual design and those obtained by automatic search. Specifically, there are seven manually designed neural network architectures, namely EfficientNet-B0[40], GoogLeNet[41], RegNet[42], ResNet-101[39], ResNeXt-101[43], Shufflenet[44], and Wide-ResNet[45]. These neural architectures are chosen for two reasons. One is that they are very popular and representative in the vision domain, and the other is that the constituent blocks of the neural architecture are extracted from these architectures, as described in Section 4.2. In the second category, we choose fifteen NAS algorithms based on different strategies to verify the superiority of GPT-NAS.

4.2 Datasets

Since there are two parts of work in this study, i.e., the implementation of the GPT-NAS and the training of the GPT model, two types of datasets are required. Firstly, image datasets are needed for training the neural architecture, so the three most popular datasets are used here, namely CIFAR-10, CIFAR-100[23], and ImageNet-1K[24]. Second, the neural architecture dataset is required in the training of the GPT model, especially in the pre-training phase, which requires a very large amount of data. Therefore, in the pre-training phase, we use NAS-Bench-101[46], while in the fine-tuning phase, the required dataset is randomly taken from the state-of-the-art neural architectures.

CIFAR-10 and CIFAR-100 are the two most popular

image classification datasets. Each contains 60 000 images, of which 50 000 are used for training, and 10 000 are used for testing. The difference between the two is the number of object classes. On CIFAR-10, 5000 images per category are used for training, while on CIFAR-100, only 500 images are used for training. Furthermore, ImageNet-1K is a more challenging dataset than the previous two, which has 1000 object classes and contains 1 281 167 training images, 50 000 validation images, and 100 000 test images. Since the image data in the test set do not give the corresponding label, only the training and validation sets are used in this experiment. In addition, it should be noted that the ImageNet-1K is too large to be realistically applied to NAS, and there is no good method to deal with it currently. So, we only take 10% of the training images to search the architecture (the validation set is consistent with the original data), but for the optimal architecture obtained from the search, we still use the whole dataset for training.

NAS-Bench-101 is a dataset of different neural architectures obtained by changing the structure of a cell in a fixed framework. Each cell has at most seven vertices and nine edges, and each neural architecture is obtained by stacking randomly composed cell structures. In the dataset, there are 423 624 neural architectures, and the corresponding performance is obtained for multiple runs on the CIFAR-10. To make the GPT model learn better neural architectures in the pre-training phase, we do not select all the neural architectures, but those with the classification accuracy of 90% or more on the validation set from them as the training data, and the final amount of neural architectures is 295 889. The dataset in the fine-tuning phase adopts the most commonly used neural architectures. In Table 2, we list the seven neural architectures and the corresponding blocks (the four blocks listed in the eighth row are those common to the neural architectures mentioned above). The number after each neural architecture in the table indicates the number of variants we can extend, depending on the properties of that neural architecture, for example, ResNet has 18 layers, 34 layers, etc. So the total number of neural architectures is 36. Note that although the number of neural architectures in the pre-training phase is 295 889 and the number of neural architectures in the fine-tuning phase is 36 when we train the GPT model, the real input data size is much larger than the number of architectures since the input data of the GPT model are layers rather than the whole architecture information. For example, a neural architecture has 20 layers, and assuming that the dimension of the input token for the GPT model is 10, the architecture can generate 10 input data.

Table 2 Neural architecture and the corresponding blocks used in fine-tuning phase.

Number	Neural architecture (Number of variants)	Block
1	EfficientNet (8)	ConvNormActivation SqueezeExcitation
2	GoogleNet (1)	Inception Avgpool
3	RegNet (14)	ResBottleneckBlock Stem
4	ResNet (5)	Bottleneck Basicblock
5	ResNext (2)	Bottleneck
6	ShuffleNet (4)	InvertedResidual
7	Wide-ResNet (2)	Bottleneck
8	Others	Maxpool BatchNormal Relu Conv

4.3 Parameters settings

The parameter settings in the experiment can be divided into two parts, one for GPT-NAS and the other for the GPT model. All experiments are performed on an Ubuntu 18.04 system with a single NVIDIA 3090 GPU with 24 GB of memory, and the code is implemented in PyTorch. In the following, we describe in detail the parameters involved in two parts.

In GPT-NAS, the two critical operations are masking network structures from each neural architecture and applying the GPT model to predict and refill the masked network structures. Therefore, we determine the initial mask rate of network structures to be 0.4 in advance in this experiment (confirmed in Section 5.2.1). Second, for the depth of neural architectures, we set the number of blocks in the range$[10, 20]$. Then, since GPT-NAS is optimized based on GA, we set the size of populations to 30, the number of iterations to 20, and the crossover and mutation rates to 0.7 and 0.5, respectively. Finally, for the neural architecture training, we make the following settings for the parameters in it. Specifically, we set the number of

epochs to 6, and use Stochastic Gradient Descent (SGD)[47] to optimize the parameters, while the learning rate is linearly incremented to 0.01. In addition, due to the different image datasets, we set the batch size differently. On CIFAR-10 or CIFAR-100, the batch size is 512, while on ImageNet-1K that is 128. When the running is finished, the neural architecture with optimal accuracy will be obtained and retrained. On CIFAR-10 and CIFAR-100, the optimal neural architecture is trained for 350 epochs, and the learning rate decays to 1/10 of the original rate every 100 epochs starting from 0.01. While on ImageNet-1K, we only train the optimal architecture for 120 epochs due to resource constraints and the learning rate decays every 30 epochs.

For the GPT model, we mostly use the same parameter settings as in the seminal paper, with a few differences as shown below. As we all know, the GPT model largely follows[48] and trains a 12-layer decoder-only transformer. However, in our experiments, since the amount of data is not as large as in the task of the seminal paper, only 4 layers of decoders are trained and only 4 attention heads are introduced in each decoder. After collation, 168 different network layer structures are finally obtained, such as convolution layer with kernel size 3, convolution layer with kernel size 5, etc. In addition, we set the input dimension to 10 and the stride to 1. Finally, we train the model for 300 epochs using the Adam optimizer with a learning rate of 1×10^{-4} and a batch size of 128.

5 Experimental Result

In this section, we discuss the experimental results in detail. The analysis of the experiments is divided into three parts, the first part is the overall performance comparison of the proposed algorithm with other state-of-the-art algorithms, the second part is the ablation experiment, and the third part is an in-depth analysis of the influence of the GPT model.

5.1 Performance overview

In this section, we describe the results of comparing GPT-NAS with other algorithms, and the specific experimental results are shown in Table 3. In the experiments, GPT-NAS is compared with four categories of related neural architectures. In addition, we also list the accuracy of the optimal neural architecture with and without the GPT model. The table shows the experimental results of different algorithms on different datasets. Two points should be noted here. The first is that there are no GPU days (a metric used to measure the time cost of NAS-related algorithms) for the neural architecture obtained by manual design, and the second is that "–" denotes null values. In addition, on CIFAR-10 and CIFAR-100, only Top1 is selected as the final accuracy due to the small number of categories, while on ImageNet-1K, both Top1 and Top5 metrics are selected as the final accuracies. The best results on each dataset have been marked in bold.

Results description. On CIFAR-10, the neural architecture obtained by GPT-NAS achieves the best result among all algorithms, with 97.69%. Compared to the manually designed neural architecture, GPT-NAS improves the classification accuracy by up to nearly 9%. Furthermore, the accuracy has increased by 4.12% compared to the ResNet-101, which is the most famous architecture today. Moreover, among the remaining architectures, the accuracy of GPT-NAS is 0.69% higher than that of EfficientNet-B0, which is the smallest accuracy difference, and 4.97%, 1.4%, 6.82%, and 1.86% higher than those of RegNet, ResNeXt-101, ShuffleNet, and Wide-ResNet, respectively. The better performance of GPT-NAS over the manually obtained neural architecture reflects that the neural architecture composed of different blocks is efficient and also demonstrates that the neural architecture learns global information. After comparing with NAS algorithms based on different strategies, it can be found that GPT-NAS also has the best performance, which is 4.61% higher than that of MetaQNN. Moreover, GPT-NAS is an algorithm based on EA, when compared with five listed state-of-the-art EA-NAS algorithms, it still outperforms more than 0.46% of them. Among all the algorithms involved in the comparison, the GO-based algorithm has the best average performance, all above 97%, but GPT-NAS still has a slight edge.

On CIFAR-100, GPT-NAS is second only to EfficientNet-B0 among all algorithms. Compared to CIFAR-10, CIFAR-100 is significantly more challenging, with only seven of all the algorithms involved in the comparison exceeding 80% in accuracy. Although the accuracy of GPT-NAS is lower than that of EfficientNet-B0, its advantage is still undeniable accuracy is compared with other algorithms. For example, it is about 20% higher than

Table 3 Experimental results of the proposed algorithm and the state-of-the-art algorithms on different datasets (the bolded indicates the optional value).

Search method	Architecture	CIFAR				ImageNet-1K			
		CIFAR-10 Top1 (%)	CIFAR-100 Top1 (%)	Number of parameters ($\times 10^6$)	GPU days	Top1 (%)	Top5 (%)	Number of parameters ($\times 10^6$)	GPU days
Human	EfficientNet-B0	97.00	**86.60**	5.3	–	77.69	93.53	5.3	–
	GoogLeNet	89.23	62.90	6.6	–	69.78	89.53	6.6	–
	RegNet	92.72	70.19	31.3	–	76.57	93.07	31.3	–
	ResNet-101	93.57	74.84	44.5	–	77.37	93.55	44.5	–
	ResNeXt-101	96.29	82.27	18.1	–	77.80	94.30	18.1	–
	ShuffleNet	90.87	77.14	3.5	–	73.70	91.09	3.5	–
	Wide-ResNet	95.83	79.50	36.5	–	78.10	93.97	68.9	–
RL	NAS-RL	96.35	–	37.4	22 400	–	–	–	–
	MetaQNN	93.08	72.86	11.2	100	–	–	–	–
	EAS	95.77	–	23.4	10	–	–	–	–
	NASNet-A	96.59	–	3.3	2000	74.00	91.60	5.3	2000
	Block-QNN-S	96.46	81.94	39.8	96	77.40	93.50	–	96
EA	Large-scale Evo	94.60	77.00	5.4/40.4	2750	–	–	–	–
	GeCNN	94.61	74.88	–	17	72.13	90.26	156.0	17
	AE-CNN	95.30	77.60	2.0/5.4	27/36	–	–	–	–
	GPCNN	94.02	–	1.7	27	–	–	–	–
	MOEA-PS	97.23	81.03	3.0/5.8	3/5	73.60	91.50	4.7	–
GO	SNAS	97.17	82.45	2.8	1.5	72.70	90.80	4.3	1.5
	P-DARTS	97.33	–	3.5	0.3	75.30	92.50	5.1	0.3
	DARTS	97.14	82.46	3.4	0.4	76.20	93.00	4.9	4.5
	ISTA-NAS	97.64	–	3.4	2.3	76.00	92.90	5.7	33.6
	U-DARTS	97.47	–	3.3	4.0	73.90	91.90	4.9	3
	FP-DARTS	97.56	–	3.8	0.04	75.30	91.80	3.8	0.04
Ours	NAS without GPT	90.77	75.20	4.6/38.1	1.5	69.53	–	104.7	4
	NAS with GPT (GPT-NAS)	**97.69**	82.81	7.1/10.5	1.5	**79.08**	**95.92**	110.9	4

Note: In Columns "Number of pamameters" and "GPU day" under "CIFAR", one value is for both CIFAR-10 and CIFAR-100, two values are for CIFAR-10 and CIFAR-100, respectively.

that of GoogLeNet. Furthermore, compared with other EA-NAS algorithms in the same category, GPT-NAS is the only one with an accuracy of more than 82%. On the other hand, regarding the number of parameters, the neural architecture obtained by searching on either CIFAR-10 or CIFAR-100 is not significantly superior. However, it is also in the middle to the upper level.

Finally, GPT-NAS outperforms all other algorithms on ImageNet-1K and is the only algorithm with a classification accuracy of over 79% on Top1 and over 95% on Top5. Since ImageNet-1K is more difficult to classify, fewer algorithms are involved in the comparison, while the classification accuracy does not differ significantly among algorithms. Except for GoogLeNet, all other algorithms have accuracies between 72% and 78% on Top1. Among the manually designed neural architectures, the optimal one is Wide-ResNet with an accuracy of 78.1%, which is 0.98% lower than GPT-NAS, while among the NAS algorithms, the optimal one is Block-QNN-S, but its accuracy is also 1.68% lower than that of GPT-NAS. The only drawback is that the number of neural architecture parameters obtained by GPT-NAS is relatively large, only less than that of GeCNN.

In addition, the time complexity (i.e., GPU days) results of the different algorithms can be observed in Table 3. First, for our method, there is no change in the time with and without the GPT. Secondly, the GPU days of 1.5 for CIFAR-10 and CIFAR-100 are significantly improved compared to many other

methods, especially the RL-based and EA-based methods. On the ImageNet dataset, our method spent 4 GPU days, which is a relatively large improvement compared to existing methods, but not particularly advantageous compared to GO-based methods. This is because most GO-based algorithms use the method of constructing a supernet, and the subsequent subnetworks implement the weight sharing, so the time consumption is significantly reduced, which is different from training all neural architectures in this paper. The results reveal the effectiveness of our proposed two acceleration strategies. Firstly, the choice to evaluate each architecture for only a small number of epochs, which, while reducing the time cost of the evaluation, significantly lowers the computational burden. Secondly, training on the predicted structures and freezing the parameters of the other structures further decline the computational expense.

In the end, by comparing the performance of the NAS in this experiment with and without the GPT model, we can find that the accuracy of the neural architecture obtained with the introduction of the GPT model is generally improved on all datasets. On the three datasets, the accuracy is improved by about 7%, 7%, and 9%, respectively, demonstrating our method's effectiveness.

Analyze and discussion. Upon closer examination, the results for GPT-NAS indicate a distinct advantage of incorporating GPT into the NAS process. The significant boost in performance on CIFAR-100 and ImageNet-1K suggests that the pre-training of the GPT model on various neural architectures can generate or predict more appropriate layers. This pre-training likely equips the GPT with an extensive 'knowledge' of architectural patterns that work well, which traditional NAS might not explore or might require significantly more time and computational resources to discover. The maintained efficiency in terms of GPU days is particularly noteworthy. It suggests that the integration of GPT does not introduce a heavier computational burden, which is often a critical concern in NAS methodologies. In other words, the GPT-NAS approach enhances the quality of the search without negatively impacting the search's efficiency. This could be due to the GPT effectively narrowing down the vast search space to more promising regions, thereby avoiding futile exploration of suboptimal architectures.

5.2 Ablation experiments

For the method proposed in this experiment, two ablation experiments are performed to verify its effectiveness. In the first part, the influence of different mask rates on the neural architecture is compared (Section 5.2.1). In the second part, we test the effectiveness of the proposed acceleration strategy (Section 5.2.2). Note that the parameter settings for the experiments in this section are slightly different from those in the main experiment (Section 5.1), as described in each subsection. In addition, ablation experiments are tested on both CIFAR-10 and CIFAR-100 datasets.

5.2.1 Validation on different mask rates

The core of this study is to mask the structures in the neural architecture effectively and to perform prediction and reconfiguration, so it is important to choose the appropriate mask rate. In this subsection, we experiment with different mask rates and choose the optimal one. For convenience, we only test the initialized individuals and do not perform genetic operations. Specifically, we choose 15 initialized neural architectures and train 90 epochs with mask rates of 0, 0.2, 0.4, 0.6, and 0.8 to test their classification accuracy. When the mask rate is 0, it means that the neural architecture has not changed its internal structure. The experimental results are shown in Table 4. In Table 4, the "mean value" indicates the average accuracy of the 15 neural architectures at the corresponding mask rates.

Results description. From Table 4, we can obtain that on CIFAR-10, the effect is the worst when the

Table 4 Experimental results of neural architecture with different mask rates. "+/=/–" indicates the number of individual neural architectures with mask rates that are better, equal, and worse in terms of classification accuracy than those without mask rates.

Dataset	Mask rate	Mean value	+/=/–
CIFAR-10	0	0.3792	–
	0.2	0.4226	10/0/5
	0.4	0.5134	15/0/0
	0.6	0.5056	15/0/0
	0.8	0.4828	13/0/2
CIFAR-100	0	0.0453	–
	0.2	0.1257	14/1/0
	0.4	0.1392	15/0/0
	0.6	0.1278	13/0/2
	0.8	0.1261	1/4/10

mask rate is 0.2, with five neural architectures being worse than the case without mask rate, and the next is when the mask rate is 0.8, with two neural architectures being worse than the case without mask rate. In addition, on CIFAR-100, only when the mask rate is 0.4, all neural architectures are better than the case without the mask rate. Furthermore, from the metric of "mean value", we can find that the average accuracy of the neural architecture on CIFAR-10 is improved by at least 5%, up to 14%, compared to the case without mask rate. While on CIFAR-100, the average accuracy of the initialized neural architecture is improved by at least 8% after the introduction of the mask rate. In summary, the neural architectures with the introduction of the mask rate have a huge average performance improvement, especially with a mask rate is 0.4. Therefore, in the main experiment, we chose the mask rate of 0.4 for the network structure as the final criterion.

Analyze and discussion. These results reveal that an appropriate mask rate helps to obtain highly accurate architectures. However, increasing the mask rate leads to diminishing returns, possibly, because the GPT model does not have enough guidance to accurately predict the masked layers. These patterns imply that GPT models, while robust in predicting architectural components, do have an uncertainty threshold. If this threshold is exceeded, their predictive ability does not translate into improved performance, emphasizing the need for strategic balance when designing masking strategies.

5.2.2 Validation on acceleration strategies

In this subsection, we implement two main types of experiments. Firstly, the neural architecture optimized by the GPT model is trained in two parts, i.e., only on the predicted blocks and on all blocks in the neural architecture, and then the two correlations are calculated. Secondly, the neural architecture is trained under different numbers of epochs and the correlation between them is calculated. Note that for correlation comparison of accuracy, we mainly use the Pearson Correlation Coefficient (PCC, between −1 and 1, the larger the value, the higher the correlation) and p-value (less than 0.05 means that they are correlated) to show.

Results description. For the first experiment, Fig. 4 shows the accuracy correlation heat map of the neural architecture for training only the blocks obtained by prediction and all blocks. The horizontal axis indicates

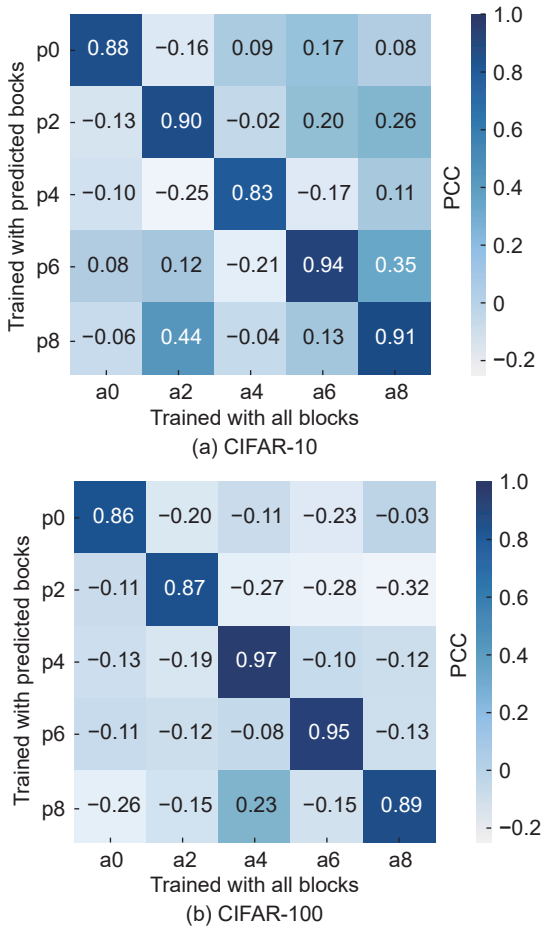


Fig. 4 Comparison of the accuracy correlation results achieved by training the blocks obtained by prediction and all blocks in the neural architecture.

that all blocks are trained, while the vertical axis indicates that the predicted blocks are trained. a0, a2, a4, a6, and a8 mean that the neural architecture is optimized with the mask rate of 0, 0.2, 0.4, 0.6, and 0.8, respectively, and all blocks are trained, while p0, p2, p4, p6, and p8 denote the neural architecture is optimized with the mask rates of 0, 0.2, 0.4, 0.6, and 0.8, respectively, and only the predicted blocks are trained. In addition, the experimental results in Fig. 4 are obtained by averaging the PCC calculated by each of the 15 neural architectures. For the second experiment, Table 5 gives the experimental results on whether there is a correlation between training only a small number of epochs versus training multiple epochs. In addition to the PCC, we also list the corresponding p-value. In addition, the first column of Table 5 indicates the comparison between different epochs, for example, "$e_6 - e_{30}$" denotes the linear correlation between the accuracy values obtained from

Table 5 Comparison of the correlation between the accuracy achieved by the neural architecture trained on a small number of epochs and multiple epochs.

Epoch pair	Dataset	PCC	p-value
$e_6 - e_{30}$	CIFAR-10	0.7247	2.68×10^{-9}
	CIFAR-100	0.8657	4.96×10^{-16}
$e_6 - e_{60}$	CIFAR-10	0.6984	1.72×10^{-8}
	CIFAR-100	0.8176	4.33×10^{-13}
$e_6 - e_{90}$	CIFAR-10	0.7046	1.13×10^{-8}
	CIFAR-100	0.8107	9.66×10^{-13}

two separately training sessions: one executed 6 epochs and the other 30 epochs.

As can be obtained from Fig. 4, the color on the diagonal in the heat map is the darkest regardless of the dataset, which means that the corresponding correlation is the highest and the values are above 80%. And the values on each diagonal line indicate the accuracy of training only the predicted blocks is relevant to training all blocks on the same neural architecture. In addition, in Table 5, we can find that the effect of using a small number of epochs to train is the same as that of using most epochs to train. It is expressed as a positive correlation on PCC, while a linear correlation between them can be proved by p-value.

Analyze and discussion. The experimental results of the two acceleration strategies can reveal that both in terms of the stability of the training and the robustness of the predicted structures, the architectures obtained by the search are satisfactory. This is attributed to the fact that we enable the GPT model to be trained on a large number of neural architectures, achieving a global understanding of the neural architecture design. Secondly, the fine-tuning of specific tasks motivate the GPT model to better adapt to the features of the tasks, resulting in further performance improvement of the searched architectures.

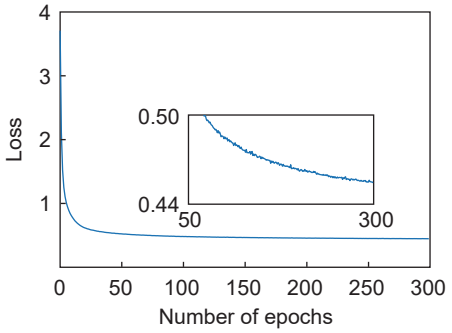
5.3 In-depth analysis

In this section, we verify the effectiveness of the GPT model, which consists of two main aspects, one is the exhibition of the training process during the pre-training and fine-tuning of the GPT model, and the other is the related ablation experiments.

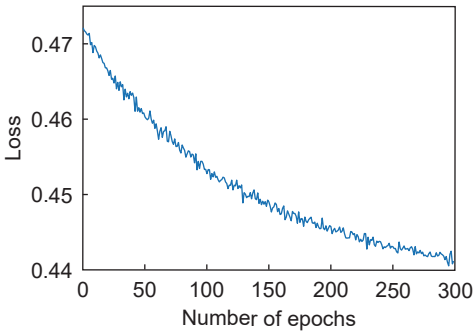
5.3.1 Training process of GPT

Figure 5 presents two line graphs depicting the loss across epochs during the pre-training and fine-tuning phases of the training process.

In Fig. 5a, labeled as the pre-training phase, we find



(a) Training process in the pre-training phase



(b) Training process in the fine-tuned phase

Fig. 5 Training process of the GPT model.

a sharp decline in loss from the start, quickly leveling off as the epochs increase. The curve starts at a loss above 3 and dramatically drops close to 0.5 within the first few epochs. This indicates an initial phase of rapid learning, where the model is capturing the fundamental patterns of the dataset (i.e., architectures). After this steep descent, the loss continues to decrease at a much slower rate, indicating that the model is starting to saturate in terms of learning from the pre-training data. An inset provides a closer look at the tail of the pre-training curve, where the loss gently slopes downwards from around 0.5, showing that the model is still making incremental improvements even as changes become more marginal.

In Fig. 5b, the graph for the fine-tuning phase starts at a loss just below 0.47, which is significantly lower than the starting point of the pre-training loss, illustrating that the model enters the fine-tuning stage with prior knowledge. The curve depicts a consistent downward trend, with the loss decreasing steadily and more linearly compared to the pre-training phase. The loss exhibits minor fluctuations but generally maintains a downward trajectory, settling just above 0.44 by the end of 300 epochs. This smooth and consistent decline suggests that the fine-tuning process is refining the parameters of the model effectively, making it more

adept at the specific task at hand.

5.3.2 Ablation experiments of GPT

(1) Consumption of search time

We test the time consumption of different search strategies. As shown in Table 6, the average time for the first two strategies (GA and GA + GPT) refers to the average time after 20 iterations, while the average search time for the last two strategies (Random and Random + GPT) is the average time per 30 individuals (since 30 individuals need to be selected for each iteration in GA).

From Table 6, we can find the GA strategy, which is a traditional approach in NAS, takes an average of 35.15 s. This approach typically involves generating a population of architectures (in this case, 30) and iteratively evolving them over several generations (20 iterations here) to find the most optimal architecture. When the GPT model is introduced into the GA (GA + GPT), the average search time increases to 56.04 s. This suggests that the integration of GPT into the GA process adds computational overhead, because the predictions of the GPT are used to optimize the architectures, which in turn requires additional computation beyond the traditional GA operations. Using the GPT model alone (Random + GPT) for NAS significantly reduces the search time to 20.05 s, nearly half the time taken by the traditional GA method. The random strategy, which serves as a baseline for comparison, has the lowest average search time of 12.36 s. This is expected as there is no computational overhead for learning or optimization; architectures are simply selected at random.

Analyzing these results, it is evident that the GPT model offers a good balance between efficiency and potentially guided exploration of the architecture space, as it does not differ much from traditional GA in terms of search time while likely providing more directed and possibly higher quality results than random search. The GA + GPT strategy, while slower than both Random + GPT and GA, offers benefits not captured by search time alone, such as potentially better optimization due to the combined exploration and exploitation mechanisms of both GA and GPT, which is demonstrated in the experiment below.

Table 6 Average search time with different strategies. Note that only the search time is recorded here, excluding the training time (because different architectures can lead to different training times).

Strategy	Time (s)
GA	35.15
GA + GPT	56.04
Random	12.36
Random + GPT	20.05

(2) Performance comparison of neural architectures with and without GPT

We randomly select 10 neural architectures and compare their accuracy on two datasets, CIFAR-10 and CIFAR-100. In Fig. 6, one line indicates the architecture chosen without using GPT for structural prediction (blue line), while the other line has GPT (orange line).

The graph for CIFAR-10 (Fig. 6a) indicates that the architectures predicted with GPT generally perform better than those selected without GPT. There is a notable variation in accuracy across different architectures, but the trend suggests that GPT provides a beneficial guide in selecting architectures that yield higher accuracy. The peaks and troughs indicate that while GPT improves performance on average, the degree of improvement varies, which might be due to the inherent variability in the architectures' suitability for CIFAR-10. For CIFAR-100, the architectures with GPT again outperform those without, although the margin of difference appears smaller compared to

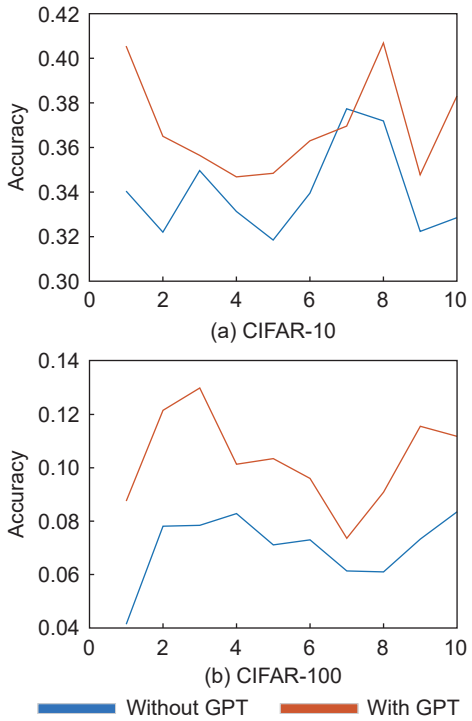


Fig. 6 Accuracy distribution of neural architectures with and without the GPT model.

CIFAR10. The variability in accuracy is also present here, with both lines showing fluctuations across the ten architectures.

Across both datasets, introducing GPT for structure prediction has a positive impact on the accuracy of the selected architectures. This implies that GPT's predictive capabilities are effective in guiding the NAS process towards more successful architectures.

(3) Comparison of search efficiency

In this experiment, we record the GPU days required to achieve different classification accuracy thresholds using the strategy combination of GA and GPT (GA + GPT) to reflect the search efficiency.

Figure 7 comprises two bar graphs representing the GPU days required to achieve specified accuracy thresholds using the GA + GPT strategy on two datasets. Each graph compares the computational time with and without GPT in the search strategy across three different accuracy thresholds.

For CIFAR-10, Fig. 7a exhibits a clear trend where the inclusion of GPT in the architecture search consistently reduces the GPU days required to reach the accuracy thresholds. At the 0.8 accuracy threshold, the "with GPT" strategy consumes slightly more than 0.3 GPU day, whereas "without GPT" consumes slightly more than half a GPU day. This disparity widens at the 0.85 threshold, where "with GPT" uses approximately 0.45 GPU days compared to "without GPT" nearly 0.8 GPU days. Figure 7b, detailing the CIFAR-100 dataset, shows a similar pattern. At a threshold of 0.55, the difference in GPU days between the two is greater than 0.2, and at a threshold of 0.65, the difference between the strategies with and without GPT is even greater.

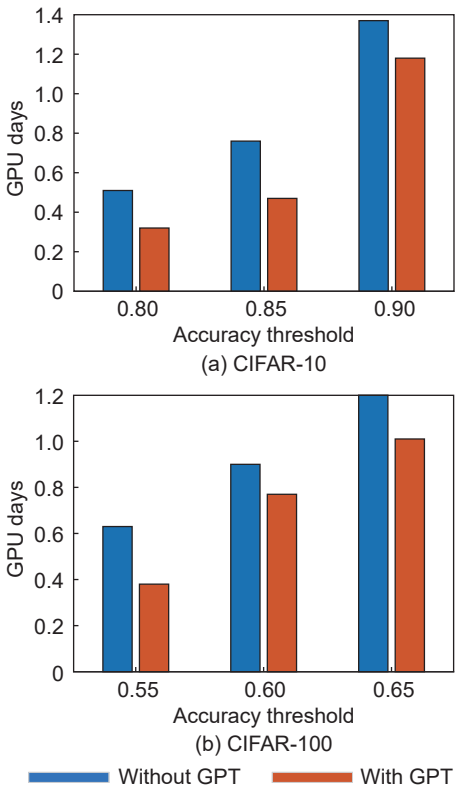


Fig. 7 GPU days required for the neural architecture to reach a certain threshold accuracy with and without the GPT model.

Figure 7 depicts that incorporating GPT into the search strategy enhances the efficiency of reaching higher accuracy levels, indicating that GPT likely guides the search towards more promising architectures. In addition, by analyzing the results, we can deduce that the GPT model provides computational savings in architecture search. This also means that with the introduction of the GPT model, the search for a better architecture becomes more efficient.

6 Conclusion and Future Work

NAS algorithms automate the design of neural networks, but traditional search strategies become overwhelmed in the face of a huge and complex search space. In this case, we propose a new approach called GPT-NAS, which utilizes the GPT model to optimize the architecture obtained from the search and compensate for the shortcomings of traditional search strategies. The proposed GPT-NAS is compared with 23 competitors on three popular datasets, where the competitors contain manually designed algorithms and NAS algorithms. The analysis of the experimental results shows that the GPT-NAS achieves state-of-the-art results and proves that the GPT model has a boosting effect on the algorithm. In future work, we will study the GPT model in more depth to make it more fully trained and to have a deeper understanding of neural architectures. In addition, we will also apply our current work to different domains, especially in the medical field. The NAS approach is designed to adapt to the domain of a particular dataset, and the dataset in the medical field meets the specificity of the dataset. By optimizing the neural structure, our GPT-NAS approach can make breakthroughs in processing medical images in a way that makes diagnosis more accurate and timely.

Acknowledgment

This work was supported by the National Nature Science Foundation of China (No. 62106161), the Fundamental Research Funds for the Central Universities (No. 1082204112364), the Sichuan University Luzhou Municipal Government Strategic Cooperation Project (No. 2022CDLZ-8), the Key R&D Program of Sichuan Province (Nos. 2022YFN0017 and 2023YFG0019), the Natural Science Foundation of Sichuan (No. 2023NSFSC0474), the Tianfiu Yongxing Laboratory Organized Research Project Funding (No. 2023CXXM14), and the Digital Media Art, Key Laboratory of Sichuan Province, Sichuan Conservatory of Music (No. 22DMAKL04).

References

[1] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, Evolving deep convolutional neural networks for image classification, *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, 2020.

[2] L. Zhang, S. Wang, F. Yuan, B. Geng, and M. Yang, Lifelong language learning with adaptive uncertainty regularization, *Inf. Sci.*, vol. 622, pp. 794–807, 2023.

[3] C. Yu, Y. Wang, C. Tang, W. Feng, and J. Lv, EU-Net: Automatic U-Net neural architecture search with differential evolutionary algorithm for medical image segmentation, *Comput. Biol. Med.*, vol. 167, p. 107579, 2023.

[4] N. S. Gupta and P. Kumar, Perspective of artificial intelligence in healthcare data management: A journey towards precision medicine, *Comput. Biol. Med.*, vol. 162, p. 107051, 2023.

[5] S. Belciug, Learning deep neural networks' architectures using differential evolution. Case study: Medical imaging processing, *Comput. Biol. Med.*, vol. 146, p. 105623, 2022.

[6] N. A. Baghdadi, A. Malki, S. F. Abdelaliem, H. M. Balaha, M. Badawy, and M. Elhosseini, An automated diagnosis and classification of COVID-19 from chest CT images using a transfer learning-based convolutional neural network, *Comput. Biol. Med.*, vol. 144, p. 105383, 2022.

[7] J. Mao, X. Yin, G. Zhang, B. Chen, Y. Chang, W. Chen, J. Yu, and Y. Wang, Pseudo-labeling generative adversarial networks for medical image classification, *Comput. Biol. Med.*, vol. 147, p. 105729, 2022.

[8] C. Tang, C. Yu, Y. Gao, J. Chen, J. Yang, J. Lang, C. Liu, L. Zhong, Z. He, and J. Lv, Deep learning in nuclear industry: A survey, *Big Data Mining and Analytics*, vol. 5, no. 2, pp. 140–160, 2022.

[9] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, LoRA: Low-rank adaptation of large language models, in *Proc. 10th Int. Conf. Learning Representations*, arXiv preprint arXiv: 2106.09685, 2022.

[10] Y. Wang, S. Agarwal, S. Mukherjee, X. Liu, J. Gao, A. H. Awadallah, and J. Gao, AdaMix: Mixture-of-adaptations for parameter-efficient model tuning, in *Proc. Conf. Empirical Methods in Natural Language Processing*, Abu Dhabi, United Arab Emirates, 2022, pp. 5744–5760.

[11] P. Ren, Y. Xiao, X. Chang, P. Y. Huang, Z. Li, X. Chen, and X. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, *ACM Comput. Surv.*, vol. 54, no. 4, p. 76, 2022.

[12] B. Zoph and Q. V. Le, Neural architecture search with reinforcement learning, in *Proc. the 5th Int. Conf. Learning Representations*, arXiv preprint arXiv: 1611.01578, 2017.

[13] B. Baker, O. Gupta, N. Naik, and R. Raskar, Designing neural network architectures using reinforcement learning, in *Proc. the 5th Int. Conf. Learning Representations*, arXiv preprint arXiv: 1611.02167, 2017.

[14] Y. Sun, X. Sun, Y. Fang, G. G. Yen, and Y. Liu, A novel training protocol for performance predictors of evolutionary neural architecture search algorithms, *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 524–536, 2021.

[15] H. Liu, K. Simonyan, and Y. Yang, DARTS: Differentiable architecture search, arXiv preprint arXiv: 1806.09055, 2018.

[16] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. L. Liu, Practical block-wise neural network architecture generation, in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018, pp. 2423–2432.

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

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, Learning transferable architectures for scalable image recognition, in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018, pp. 8697–8710.

[19] L. Xie and A. Yuille, Genetic CNN, in *Proc. IEEE Int. Conf. Computer Vision*, Venice, Italy, 2017, pp. 1388–1397.

[20] M. Suganuma, S. Shirakawa, and T. Nagao, A genetic programming approach to designing convolutional neural network architectures, in *Proc. Genetic and Evolutionary Computation Conf.*, Berlin, Germany, 2017, pp. 497–504.

[21] T. Zhang, C. Lei, Z. Zhang, X. B. Meng, and C. L. P. Chen, AS-NAS: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning, *IEEE Trans. Evol. Comput.*, vol. 25, no. 5, pp. 830–841, 2021.

[22] K. Maziarz, M. Tan, A. Khorlin, M. Georgiev, and A. Gesmundo, Evolutionary-neural hybrid agents for architecture search, arXiv preprint arXiv: 1811.09828, 2018.

[23] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf, 2023.

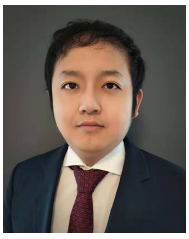
[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., ImageNet large scale visual recognition challenge, *Int.*

J. Comput. Vis., vol. 115, no. 3, pp. 211–252, 2015.
[25] S. Lu, Y. Wang, P. Yao, C. Li, J. Tan, F. Deng, X. Wang, and C. Song, Conformer space neural architecture search for multi-task audio separation, in *Proc. 23rd Annu. Conf. Int. Speech Communication Association*, Incheon, Korea, 2022, pp. 5358–5362.
[26] L. Pouy, F. Khenfri, P. Leserf, C. Mhraida, and C. Larouci, Open-NAS: A customizable search space for neural architecture search, in *Proc. 8th Int. Conf. Machine Learning Technologies*, Stockholm, Sweden, 2023, pp. 102-107.
[27] C. Jin, J. Huang, T. Wei, and Y. Chen, Neural architecture search based on dual attention mechanism for image classification, *Math. Biosci. Eng.*, vol. 20, no. 2, pp. 2691–2715, 2023.
[28] H. Nori, N. King, S. M. McKinney, D. Carignan, and E. Horvitz, Capabilities of GPT-4 on medical challenge problems, arXiv preprint arXiv: 2303.13375, 2023.
[29] W. Hou and Z. Ji, GeneTuring tests GPT models in genomics, bioRxiv, doi: 10.1101/2023.03.11.532238.
[30] N. Nascimento, C. Tavares, P. Alencar, and D. Cowan, GPT in data science: A practical exploration of model selection, in *Proc. 2023 IEEE Int. Conf. Big Data*, Sorrento, Italy, 2023, pp. 4325–4334.
[31] Z. Hu, Y. Dong, K. Wang, K. W. Chang, and Y. Sun, GPT-GNN: Generative pre-training of graph neural networks, in *Proc. 26th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, doi: 10.1145/3394486.3403237.
[32] M. Zheng, X. Su, S. You, F. Wang, C. Qian, C. Xu, and S. Albanie, Can GPT-4 perform neural architecture search? arXiv preprint arXiv: 2304.10970, 2023.
[33] H. Wang, Y. Gao, X. Zheng, P. Zhang, H. Chen, J. Bu, and P. S. Yu, Graph neural architecture search with GPT-4, arXiv preprint arXiv: 2310.01436, 2023.
[34] Y. Sun, G. G. Yen, B. Xue, M. Zhang, and J. Lv, ArcText: A unified text approach to describing convolutional neural network architectures, *IEEE Trans. Artif. Intell.*, vol. 3, no. 4, pp. 526–540, 2022.
[35] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, Completely automated CNN architecture design based on blocks, *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, 2020.
[36] J. Mellor, J. Turner, A. J. Storkey, and E. J. Crowley, Neural architecture search without training, in *Proc. 38th Int. Conf. Machine Learning*, https://proceedings.mlr.press/v139/mellor21a.html, 2023.
[37] B. Chen, P. Li, B. Li, C. Lin, C. Li, M. Sun, J. Yan, and W. Ouyang, BN-NAS: Neural architecture search with batch normalization, in *Proc. IEEE/CVF Int. Conf. Computer Vision*, Montreal, Canada, 2021, pp. 307–316.
[38] B. Baker, O. Gupta, R. Raskar, and N. Naik, Accelerating neural architecture search using performance prediction, in *Proc. Int. Conf. Learning Representations*, arXiv preprint arXiv: 1705.10823, 2018.
[39] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 770–778.
[40] M. Tan and Q. V. Le, EfficientNet: Rethinking model scaling for convolutional neural networks, in *Proc. 36th Int. Conf. Machine Learning*, Long Beach, CA, USA, 2019, pp. 6105–6114.
[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Going deeper with convolutions, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Boston, MA, USA, 2015, pp. 1–9.
[42] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, Designing network design spaces, in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, Seattle, WA, USA, 2020, pp. 10425–10433.
[43] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, Aggregated residual transformations for deep neural networks, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017, pp. 5987–5995.
[44] N. Ma, X. Zhang, H. T. Zheng, and J. Sun, ShuffleNet V2: Practical guidelines for efficient CNN architecture design, in *Proc. 15th European Conf. Computer Vision*, Munich, Germany, 2018, pp. 122–138.
[45] S. Zagoruyko and N. Komodakis, Wide residual networks, in *Proc. the British Machine Vision Conf.*, arXiv preprint arXiv: 1605.07146, 2016.
[46] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, NAS-bench-101: Towards reproducible neural architecture search, in *Proc. 36th Int. Conf. Machine Learning*, Long Beach, CA, USA, 2019, pp. 7105–7114.
[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, Attention is all you need, in *Proc. 31st Int. Conf. Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 6000–6010.

Caiyang Yu received the MEng degree in computer architecture from Wenzhou University, China in 2018. He is currently a PhD candidate at College of Computer Science, Sichuan University, China. His research focuses on the neural network search and evolutionary computation.

Xianggen Liu received the PhD degree from Tsinghua University, China in 2021. He is currently an associate professor at College of Computer Science, Sichuan University, China. His current research interests include machine learning and nature language processing.

Yifan Wang received the BEng degree from Liaoning University, China in 2023. She is currently a master student at College of Computer Science, Sichuan University, China. Her research focuses on computer vision and object detection.



Wentao Feng received the PhD degree in geothermal engineering from Clausthal University of Technology (TU Clausthal), Clausthal-Zellerfeld, Germany. He is currently a postdoctoral researcher at College of Computer Science, Sichuan University, China. His research interests include artificial intelligence, numerical simulation, and smart energy.

Jiancheng Lv received the PhD degree in computer science and engineering from University of Electronic Science and Technology of China in 2006. He was a research fellow at Department of Electrical and Computer Engineering, National University of Singapore, Singapore. He is currently a professor at College of Computer Science, Sichuan University, China. His research interests include neural networks, machine learning, and big data.

Yun Liu received the MEng degree from Wenzhou University, China in 2022. She is currently a PhD candidate at College of Computer Science, Sichuan University, China. Her research focuses on intelligent manufacturing.

Xiong Deng received the MEng degree in computer science from Stevens Institute of Technology, NJ, USA in 2022, and the MEng degree in mechanical engineering from Stevens Institute of Technology, NJ, USA in 2017, where he serves as both a teaching assistant and research assistant at Department of Mechanical Engineering. His research interests encompass artificial intelligence, big data, cloud computing, database, robotics, soft materials, and additive manufacturing.

Chenwei Tang received the PhD degree in computer science and technology from Sichuan University, China in 2020. She is currently an associate professor at College of Computer Science, Sichuan University, China. Her research focuses on the neural network methods for zero-shot learning, and industrial intelligence.