Machine Learning (2023) 112:1315–1335
https://doi.org/10.1007/s10994-022-06287-5

Neural predictor-based automated graph classifier framework

Babatounde Moctard Oloulade[1] · Jianliang Gao[1] · Jiamin Chen[1] · Raeed Al-Sabri[1] · Tengfei Lyu[1]

Received: 15 February 2022 / Revised: 31 October 2022 / Accepted: 23 November 2022 /
Published online: 20 December 2022

Abstract
Graph Neural Architecture Search (Graph-NAS) methods have shown great potential in finding better graph neural network designs compared to handcrafted designs. However, existing Graph-NAS frameworks are based on complex algorithms and fail to maintain low costs for high scalability with high performance. They require full training of thousands of graph neural networks to inform the search process, resulting in a prohibitive computational cost, which is not necessarily affordable for the users interested. Due to the computation cost, many researchers have limited the search space exploration ability, which may lead to a local optimum solution. In this paper, we propose a performance predictor-based graph neural architecture search (PGNAS) framework. The proposed approach consists of three conceptually much simpler and basic phases, and can broadly explore a search space with a much cheaper computation cost. We train n sampled architectures from a search space to generate n (architecture, validation accuracy) pairs used to train a performance distributions learner where the features are represented by the architecture description and the validation accuracy denotes the target. Next, we use this performance distribution learner to predict the validation accuracies of architectures in the search space. Finally, we train the top-K predicted architectures and choose the architecture with the best validation result. Although our approach seems simple, it is efficient and scalable; experiment results show that PGNAS outperforms existing both handcrafted and Graph-NAS models on four benchmark datasets.

Keywords Graph classification · Neural architecture search · Neural performance predictor · Graph neural network

Editors: Krzysztof Dembczynski and Emilie Devijver.

✉ Jianliang Gao
gaojianliang@csu.edu.cn

Extended author information available on the last page of the article

1 Introduction

Graph neural networks (GNNs) have become a powerful approach for graph data processing as they achieve great performances in various tasks (Wu et al., 2021; Liu et al., 2020). GNN models are usually handcraft-built models. However, a handcraft-built GNN model is time-consuming and requires expert experience and unwritten rules of thumb because of the availability of multiple-choice combinations for the selection of GNN components, which are sensitive to variation. The complexity of GNN model architectures has brought significant challenges to the existing GNN efficiencies. As response, many studies have attempted to apply neural architecture search(NAS) approaches to graph representation learning. Generally, NAS frameworks work in two iterative stages. The first step is to generate a child model from a search space, and the second step is to evaluate it. The evaluation in each iteration serves as a benchmark for the comparison in the following iterations. The difference between the different frameworks, therefore, lies in three essential points, which are: (1) how to design the search space, (2) how to evaluate a generated architecture, and (3) how to optimize the search for finding the best architecture efficiently. The above challenges are commonly solved through three main panels: search space, search strategy, and optimization strategy (Oloulade et al., 2021).

Message-passing schemes have been given much credit by researchers for their ability to better represent nodes' properties. The attention function, the aggregation function, the attention head, and others are all components of this representation that can be learned. Several possibilities exist for each of these functions, which have a significant impact on the model's performance. There are two broad categories of search space used in existing studies to discover the best combination of components. The first class defines a search space for the various functions but uses fixed hyper-parameters (Zhou et al., 2019; Zhao et al., 2020) while the second class defines a search space including functions and hyper-parameters (Li & King, 2020). While the latter will lead to a more stable optimum solution, the search space scale might have supplementary computation costs. It is worth noting that exploring all possible GNNs architectures in the vast search space is too time-consuming or impossible for big graph data. This feature poses a scalability issue when searching for the optimal model. Thus, the main challenge is how to find a trade-off between high performance, low cost, and strengthened productivity through the search algorithm. Several search algorithms have been proposed for different frameworks, including reinforcement learning (RL) (Gao et al., 2020), genetic algorithm (GA) (Zhou et al., 2019), bayesian optimization (BO) (Yoon et al., 2020), differentiable search (DS) (Cai et al., 2021), and random search (RS) (You et al., 2020). These search strategies have also been combined (Zhou et al., 2019).

While these frameworks have proven to outperform handcrafted GNN models, it is still necessary to find a robust trade-off between performance, cost, and scalability. Existing solutions fail to maintain low costs for high scalability and high performance (Zhang et al., 2021; Oloulade et al., 2021). Existing methods have to limit search space exploration because of the expensive computation cost. The main challenge remains how to speed up the child model evaluation and improve the efficiency of the evaluation process. To overcome this challenge, several strategies have been proposed and have been applied individually and in combination. The first strategy that quickly gained consensus is the weight sharing, which helps prevent a newly generated child model from being trained from scratch to convergence. However, Zhou et al. (2019) proved through experiments that weight sharing is not empirically helpful. Another strategy is the single-path one-shot

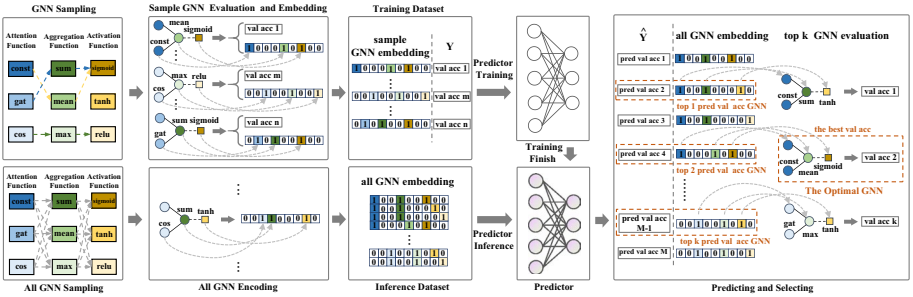


Fig. 1 Predictor-based graph neural architecture search framework. First, a neural predictor is built using encoded few sampled GNN configurations from the search space and their validation accuracy distribution. Next, the predictor is used to predict all GNN configurations in the search space

model proposed (Guo et al., 2020) which uses uniform sampling with only one operation activated between the input and output pair at each iteration. More recently, GraphPAS (Chen et al., 2021) proposed parallel computation with a genetic algorithm. However, existing frameworks still need to train several GNN configurations, resulting in a prohibitive computational cost, which is not necessarily affordable for the users interested. Recently, the predictor-based NAS approach (White et al., 2021, 2021), also called surrogate-based modeling (Search, 2020; Shi et al., 2020, 2021) has been successfully used for convolutional neural networks (CNNs) architectures search. However, applying NAS algorithms to find GNN architectures is not trivial due to two major challenges as follows: (1) The search space of GNN architectures is different from the search space of CNN architecture. For example, only the kernel size needed to be specified for convolution operation in CNN while in GNN, the message passing-based convolution is characterized by a sequence of operations including at least an aggregation function and a combination operation, which makes the architecture encoding construction non-trivial. (2) CNNs are predominantly applied to data represented as a regular grid in the Euclidean space and fail to extract latent representations from graph data. This drawback is due to graphs being non-Euclidean data, which cannot be represented in an n-dimensional linear space. Thus, some important operations, such as convolutions computed in Euclidean space with CNN, are difficult to implement for graph data. Moreover existing NAS methods for CNN commonly only search for operations while in this work we consider both operations and hyperparameters simultaneously. We propose a simple but efficient Neural predictor for graph neural architecture search (PGNAS) framework to alleviate the above problems. In the proposed method, we first build a performance distribution learner by training n random GNN configurations uniformly sampled from the search space and recording their performance on a validation dataset. Encoding each architecture and obtaining pairs with n (encoded architecture, validation accuracy). Training the performance distributions learner using these data. Next, we predict the performance of GNN configurations in the search space using the performance distributions learner and select the $top-k$ promising GNN configurations for the final step. Finally, we get the best configuration from the $top-k$ by training them and selecting the GNN's configuration with the highest validation accuracy. The workflow of our framework is illustrated in Figure 1. We use parallel computation to speed up the first step, a regression problem where we first generate a dataset of n samples to train on. The second step can be carried out efficiently as predicting a model's performance is cheap. The last step is a standard validation where we only evaluate a

well-curated set of k GNN configurations. While the method outlined above might seem straightforward, it is very effective. Our contributions are summarized as follows:

- To the best of our knowledge, we make the first attempt to use a performance neural predictor-based framework for graph neural architecture search solving the problem of search efficiency.
- We propose an efficient performance predictor-based search algorithm PGNAS for graph neural network architectures search. In PGNAS, we simultaneously consider GNN architecture components, hyperparameters, and feature engineering. We predict the performance of sampled GNN configurations, significantly reducing GNN configurations' evaluation cost compared with existing frameworks. Moreover, we use the simple yet effective ranking process to choose the best model to deploy.
- We discuss PGNAS scalability, and we evaluate it by conducting extensive experiments on different types of datasets compared with existing both handcrafted and automated graph neural architecture search frameworks. Furthermore, experiment results show that PGNAS can significantly accelerate the graph neural architecture search framework while improving efficiency.

2 Related work

Existing Graph-NAS frameworks generally consist of a recursive process involving three components: a search space, a search strategy, and a performance estimation strategy. The search strategy generates architectures from a predefined search space over time based on the performance evaluation determined via the estimation strategy. Many methods based on reinforcement learning, genetic algorithm, Bayesian optimization, differentiable search, random search, or a combination of them have been used for building the Graph-NAS framework. The reinforcement learning-based method uses a recurrent network as a controller to generate child models from a search and the reward of the child model is computed as feedback to optimize the expected performance of generated child models. This approach has been used by many works with some differences. For example, in GraphNAS (Gao et al., 2020) a generated child model by the controller is replaced in its entirety with a newly generated child model regardless of their similarities while AGNN (Zhou et al., 2019) uses a controller-based reinforce rule of policy gradient (Sutton et al., 1999) to gradually validate models with small steps. Moreover, GraphNAS adopted the standard parameter sharing strategy while AGNN uses constrained parameter sharing using three constraints. It is worth noting that in the former methods it is hard for the controller to learn which part of a model influences the accuracy change compared to another model and standard parameter sharing would lead to unstable training when sharing parameters among heterogeneous GNNs.

Genetic algorithm-based search has been used by many works (Shi et al., 2020; Li & King, 2020; Chen et al., 2021), in which the population is represented by the search space and an individual is represented by a GNN architecture. The main limitation of this approach is the slow convergence (Oloulade et al., 2021). To overcome these burdens, GraphPAS (Chen et al., 2021) proposes a parallel genetic algorithm-based framework designing a parallel sharing-based evolution learning, which improves search efficiency.

In a differentiable search for Graph-NAS, the key concept is to build a supernet by stacking mixed operations. The output of each layer is the combination of applied mixed

operations. The coefficients represent selection parameters and the loss function is minimized for an architectural search. Operations with a higher coefficient in each layer are chosen to create the final GNN architecture. In existing such Graph-NAS frameworks, a stack is usually defined as a direct acyclic graph (Zhao et al., 2021) involving an ordered series of nodes. The continuous relaxation scheme depends on a parameterization trick, which is less noisy and easier to train compared to the previous search algorithm. Although this search approach can outperform the previous search approaches in terms of speed and search quality, the construction of the supernet makes it computationally expensive costs and it requires a differentiable latency approximation (Oloulade et al., 2021).

3 Proposed framework

In this section, we first formulate the graph neural architecture search problem and present the predictor-based framework. Finally, we discuss the predictor-based search framework.

3.1 Problem formulation

Considering a graph $G = (V, E)$ where V and E represent a set of nodes and a set of edges, respectively. The neural network notion of a "layer of neurons" in GNNs is translated into a composition of functions: the first aggregates information from the neighborhood of each node $N(i)$ forming an intermediate vector $h_{N(i)}$ (AGG); the second combines this intermediate vector with the current node representation h_i ($COMB$); and applies a normalization function ($NORM$) and dropout (∂) before applying an activation function (σ). The following equations describe this process:

$$h^k_{N(i)} = AGG(h^{k-1}_j : j \in N(i)) \tag{1}$$

$$h^k_i = \sigma(\partial(NORM(COMB(h^{(k-1)}_i, h^k_{N(i)})))) \tag{2}$$

The node's feature vector is conventionally used as the first hidden representation, $h^{(0)}_i = x_i$ where x_i represents the feature the node i (Kipf & Welling, 2017).

Given how a GNN works, the problem of searching for its architecture can be formally defined as follows: for a search space S of graph neural networks, a training dataset D^{train}, a validation dataset D^{val} and performance evaluation Y, the goal is to find the optimal GNN's configuration $gnn_{opt} \in S$ with the best-expected performance evaluation Y on D^{val}, when its parameters θ^* are set by minimizing a loss function L on D^{train}, fixing the following bi-level optimization problem mathematically expressed as follows:

$$\begin{aligned} gnn^* &= \underset{s \in S}{argmax}\ Y(gnn(\theta^*), D_{val}) \\ s.t.\ \theta^* &= \underset{\theta}{argmin}\ L(gnn(\theta), D_{train}) \end{aligned} \tag{3}$$

3.2 Proposed search framework

Our goal is to model the validation accuracy Y of a graph neural network (GNN) configuration $gnn \in S$ using the description of the GNN, which is given by a one-hot-based encoder

Table 1 GNN design space

Components	Choices
Aggregation function	Sum, max ,mean
Convolution function	Linear, GCNConv, GENConv, LEConv
Dropout	False, 0.3, 0.6
Activation function	ReLU6, Tanh, ELU, PReLU, Sigmoid
Normalizer	BatchNorm, GraphNorm, InstanceNorm
Hidden dimension	64, 128,256
Multi-head	1,2,4,8
Learning rate	1e-2, 1e-3, 5e-3, 5e-4
Weight regulation	0, 1e-3, 5e-4
Optimizer	adam, sgd
Loss criterion	CrossEntropyLoss, MultiMarginLoss
Pooling type	global_add_pool, global_max_pool

model $\xi(\cdot)$. For each GNN configuration *gnn* trained, we record the encoding of the configuration and its final validation accuracy. We sample and train a population of n configurations obtaining a set $S_p = \{(\xi(gnn_1), Y_1), (\xi(gnn_2), Y_2), ..., (\xi(gnn_n), Y_n)\}$. The obtained set is used to train a performance distribution learner and further used to predict the validation accuracy of configurations in the search space. The encoded configuration of top-k most promising predicted GNN configurations are decoded and the corresponding models are trained and validated. Next, the best GNN configuration is chosen based on the validation accuracy. As we shall see in the subsequent section, predicting GNN configuration performance is a better way to improve search efficiency as the performance distribution learner provides a curated list of promising GNN configurations after broadly evaluating the configurations in the search space. Thus, avoiding the local optimum problem. Figure 1 shows the predictor-based framework, which we propose to solve Eq. 3.

3.2.1 GNN design space

A GNN architecture consists of a combination of many components. Different components can take different values called options. We depict a general design space of GNNs that consider GNN architecture components such as aggregation function and pooling function and hyperparameters components such as learning rate and dropout. The designed search space is presented in Table 1. It consists of 12 design components, resulting in over 33.10^6 possible architectures with two layers. Our objective in proposing such is to show how focusing on the design space can improve GNN research and how insensitive our framework is to the search space size. We emphasize that the design space is easily extendable to new design dimensions that emerge in state-of-the-art models.

3.2.2 Sampling and encoding methods

Due to the vast size of the search space, a non-representative sample of the search space would be inappropriate for generalizing the predictor to the whole search space. To overcome this challenge, we propose a controlled stratified random sampling method. For each option of each component in the search space, we use a proportional stratified sampling method to sample s GNN configurations from the search space. Then, we force

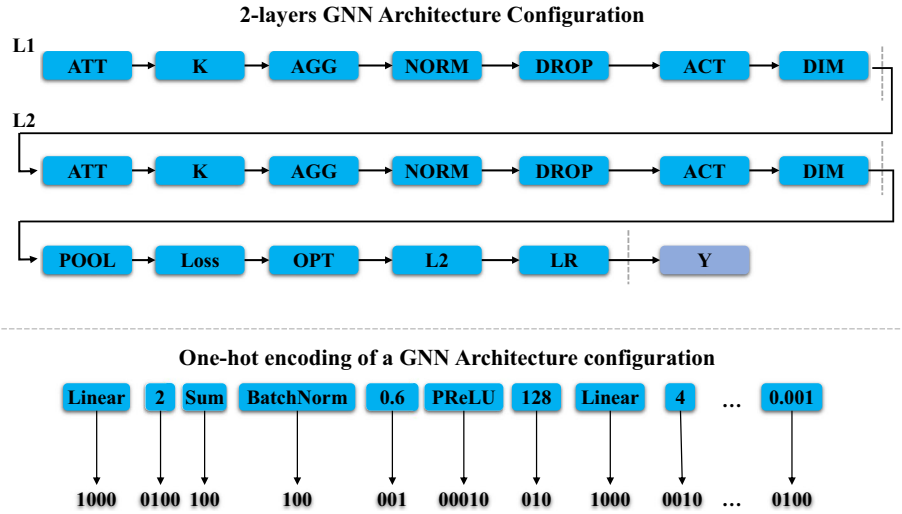


Fig. 2 GNN architecture and its encoding representation

the value of the current function to be the value of the current option in the loop. The total number of sampled GNN configurations is $n = s \times T_{opt}$, where T_{opt} is the total number of options. The algorithm of the proposed sampling method is shown in Algorithm 1. Furthermore, the ablation study result presented in Sect. 4.8 shows that the proposed sampling method is effective compared with the random sampling method.

Upon the search space being built, n architectures were sampled from the search space using the sampling the aforementioned method. We transform the list of chosen options into a one-hot encoded sparse vector using the one-hot encoding of size equal to the total number of options in the search space. One-hot encoding is often used in machine learning algorithms for converting categorical features into numerical ones and is commonly used as a first step to more sophisticated representation methods (Hancock & Khoshgoftaar, 2020). For each selected option, we use the one-hot encoding of size p, where p is the number of choices available for the corresponding function in the search space. Finally, we concatenate the encoding of each option to get the encoding of a GNN's configuration. An abstract example of the encoding method is shown in Fig. 2. The built dataset is used to train a performance distribution learner.

Algorithm 1 Controlled stratified random sampling algorithm

Input:
A search space SS: *dictionary*
Sample size n : *integer*
Output:
sampled GNN configurations: list of dictionary

$q \leftarrow Total\ choices\ in\ SS$
$TempDict = SS$
Let $ModelList \leftarrow []$. # empty list
while length of $ModelList < (n \backslash q) + 1$ **do**
    Let $ModelDict \leftarrow \{\}$. # empty dictionary
    for function and options in SS **do**
        if $TempDict[function]$ is Empty **then**
            $TempDict[function] \leftarrow options$
        end if
        $ModelDict[function] \leftarrow$ Random ($TempDict[function]$)
        Remove selected option from $TempDict[function]$
    end for
    if $ModelDict$ not in $ModelList$ **then**
        $ModelList.append$(ModelDict)
    end if
end while
$samples\_list \leftarrow []$
for $function, option \in$SearchSpace.$items()$ **do**
    for $component \in option.keys()$ **do**
        for $submodel \in ModelList$ **do**
            $model_{temp} \leftarrow submodel$
            $model_{temp}[function] \leftarrow option[component]$
            $samples\_list.append(model_{temp})$
        end for
    end for
end for
return $samples\_list[: n]$

4 Experiments

We apply our method to find the optimal GNN model given the graph classification task to answer the following four questions:

- How does GNN models found by PGNAS compare with state-of-the-art handcrafted models and the ones searched by other NAS methods?

Table 2 Benchmark datasets.

Dataset	NG	AN	NF	NC	Field	Source
enzymes	600	32.63	21	6	Chemical dataset	(Borgwardt et al., 2005; Schomburg et al., 2004)
imdb-b	1000	19.77	–	2	Social dataset	(Yanardag & Vishwanathan, 2015)
proteins	1113	39.06	4	2	Chemical dataset	(Borgwardt et al., 2005; Dobson & Doig, 2003)
dd	1178	284.32	–	2	Chemical dataset	(Dobson & Doig, 2003; Shervashidze et al., 2011)
bzr	405	35.75	3	2	Chemical dataset	(Sutherland et al., 2003)

NG Number of Graphs; *AN* Average Nodes; *NF* Node features; *NC* Number of Classes

- How does the search efficiency of PGNAS compare with those of different search methods?
- Whether or not the model encoding strategy effectively helps the predictor GNN's configuration to learn model performance distribution?
- How well does the neural predictor model learn GNN configurations performance distribution?
- How does the proposed sampling method perform compared with random sampling?

This section presents more details about the datasets, baseline methods, experimental settings, and results. We conduct extensive performance, efficiency, and analysis experiments to evaluate the effectiveness and the scalability of PGNAS.

4.1 Datasets

We use four benchmark datasets in this work, including the proteins dataset, enzymes dataset, imdb-b dataset, and dd datasets. The statistics of dataset characteristics are summarized in Table 2.

4.2 Baseline methods

We compare GNNs architecture identified by our model against baselines, including the state-of-art handcrafted architectures such as WEGL (Kolouri et al., 2021), BASGCN (Bai et al., 2020), DiffPool (Ying et al., 2018), CurGraph (Wang et al., 2021), PGCN (Pasa et al., 2021) and NAS methods such as GraphNAS (Gao et al., 2020), Genetic-GNN (Shi et al., 2020), and Auto-GNAS (Chen et al., 2022).

4.3 Experiment settings

The GNN architectures designed by PGNAS are implemented by the Pytorch-Geometric library(Fey & Lenssen, 2019). We evaluate our model on graph classification tasks with inductive settings. For all experiments, we use a consistent setup, where random 80%/10%/10% train/val/test splits are used for dataset split, we use the accuracy metric for GNN performance evaluation. The total number of sampled configurations for each function in the search space s is set to 12, and the total number of sampled GNN configurations is given by $n = s \times n_0$ where n_0 is the total number of choices in the search

space, which is 50. The number of times z_{init} a sampled GNN's configuration should be trained before recording its performance in the first step is set to 1, the number of best-predicted GNN configurations k to keep for final evaluation is set to 20, the number of time z_{final} the predicted best GNN configurations are trained before ranking their average performance is set to 5, and the number of time t the best GNN's configuration is tested before we report its average performance is set to 10. We set the number of epochs to 200 for all experiments.

4.4 Performance distribution learner evaluation

For evaluating the ability of the performance distribution learner to learn the sampled model performances' distribution, we evaluate the predictor's generalization ability to the whole search space. We use R^2 metric for both evaluations. Figure 3 shows the power of the performance distribution learner to learn graph neural network performance distribution.

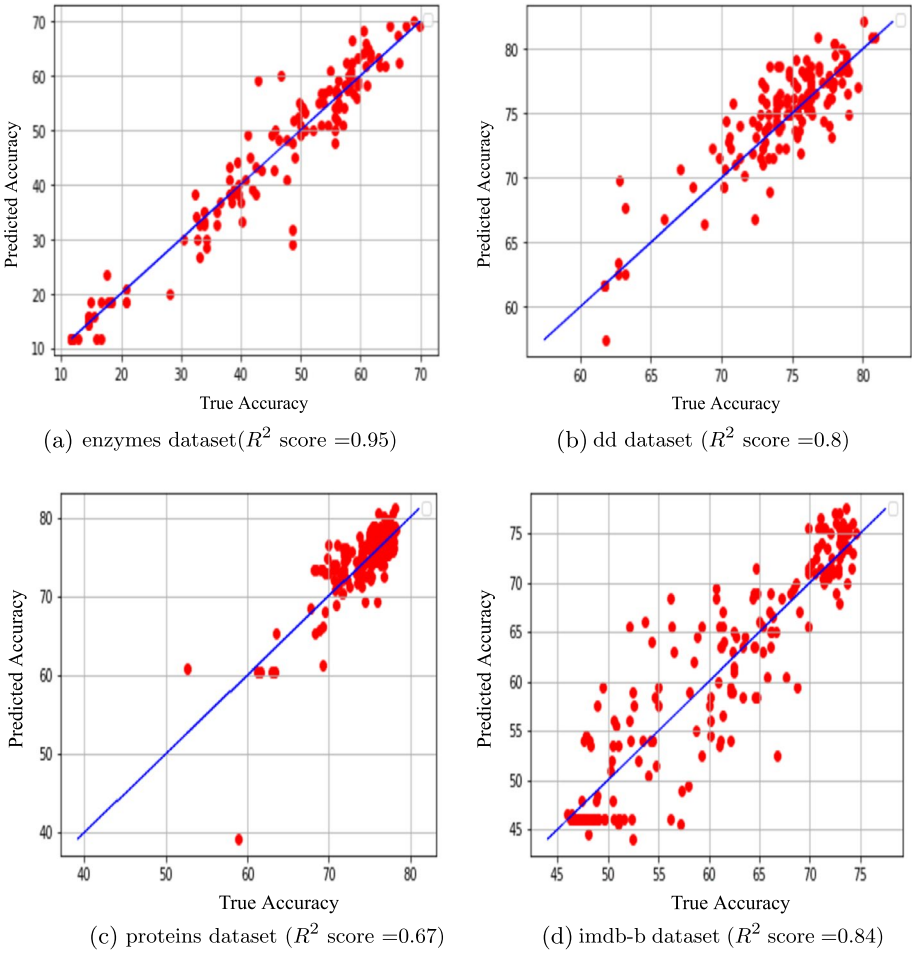
4.5 Evaluation on graph classification task

To validate the performance of our model, we compare the GNN models discovered by PGNAS with handcrafted models and those designed by other search approaches. The performance of the graph classification task is summarized in Table 3. We report the test accuracy of baseline handcrafted models from the original work. Results of NAS-based baseline models are replicated using the Auto-GNAS[1] framework with the search space proposed by GraphNAS (Gao et al., 2020). For a fair comparison, we reduce the search space defined in 3.2.1 to the same search space. We use "global_add_pooling" operation for graph pooling. The best result for each group of our baselines is underlined, and the best result on each dataset is in boldface. Looking only at the handcrafted methods, we notice that there is no clear winner over the considered datasets. For example, WELG is the best model for the proteins dataset while PGCN is the best model for the enzymes dataset and BASGCN achieves the best performance on the dd dataset. Considering that these datasets are from different fields, it shows the need for adaptive graph neural architecture for graph classification. Besides, the result indicates that PGNAS can achieve competitive results on all four datasets, which demonstrates the effectiveness of PGNAS in searching efficiently neural architecture for graph classification and proving its scalability ability to different types of networks.

4.6 Efficiency of PGNAS

To show that PGNAS is efficient and effective, we do a supplementary experiment with different values for n where we use random search instead of our search algorithm and note this framework as *PGNAS_RSe*. Since the total number n_{total} of trained and validated GNN configurations by PGNAS in the whole search process is $n_{total} = n * z_{init} + k * z_{final}$, we train and validate n_{total} GNN configurations for a fair comparison. As shown in Fig. 4, PGNAS significantly outperforms random search. Meanwhile, it is shown that when the number of samples increases, the performance

[1] https://github.com/AutoMachine0/Auto-GNAS



(a) enzymes dataset(R^2 score =0.95) (b) dd dataset (R^2 score =0.8)

(c) proteins dataset (R^2 score =0.67) (d) imdb-b dataset (R^2 score =0.84)

Fig. 3 The ability of the neural predictor to learn graph neural network performance distributions

of PGNAS increases, but PGNAS can already achieve very competitive results with 600 GNN configuration samples. For simplicity, we make the experiment only on two datasets, the proteins dataset, and the dd dataset . It is worth noting that existing NAS methods trained a larger number of models before stopping exploring search space. For example, GraphNAS (Gao et al., 2020) and GraphPAS (Chen et al., 2021) stop the search process after training 2000 models. However, even if this number is larger, it still limits the search efficiency because it represents only 0.2% of the search space of GraphNAS and GraphPAS, which limits the search space exploration and leads to a sub-optimal solution. GPNAS can train only 600 models for a whole search process and perform better because of its huge search space exploration capability.

Table 3 Performance comparison between PGNAS and other methods on four datasets

	Models	Proteins dataset	Enzymes dataset	imdb-b dataset	dd dataset
Handcrafted Models	WEGL (Kolouri et al., 2021)	76.5 ± 4.2	60.5 ± 5.9	75.4 ± 5.0	–
	BASGCN (Bai et al., 2020)	76.5 ± 0.59	–	73.86 ± 0.92	80.71 ± 0.99
	DiffPool (Ying et al., 2018)	76.25	62.53	–	80.64
	CurGraph (Wang et al., 2021)	75.4 ± 3.1	64.8 ± 3.3	73.4 ± 3.2	78.6 ± 3.0
	PGCN (Pasa et al., 2021)	75.31 ± 0.31	70.5 ± 1.77	72.60 ± 3.80	79.45 ± 0.29
NAS Models	GraphNAS (Gao et al., 2020)	75.20 ± 0.025	–	73.60 ± 0.046	76.74 ± 0.045
	SNAG (Zhao et al., 2020)	70.53±0.0311	–	72.50±0.046	72.23 ±0.038
	PAS (Chen et al., 2022)	76.64 ± 0.032	–	75.10 ± 0.053	78.96±0.0336
	PGNAS_baseline	76.82 ±2.48	70.83 ± 2.96	75.12 ±1.81	80.63 ± 1.71
	PGNAS	77.56 ± 1.1	71.13 ± 4.8	75.18 ±3.2	82.86 ± 2.77

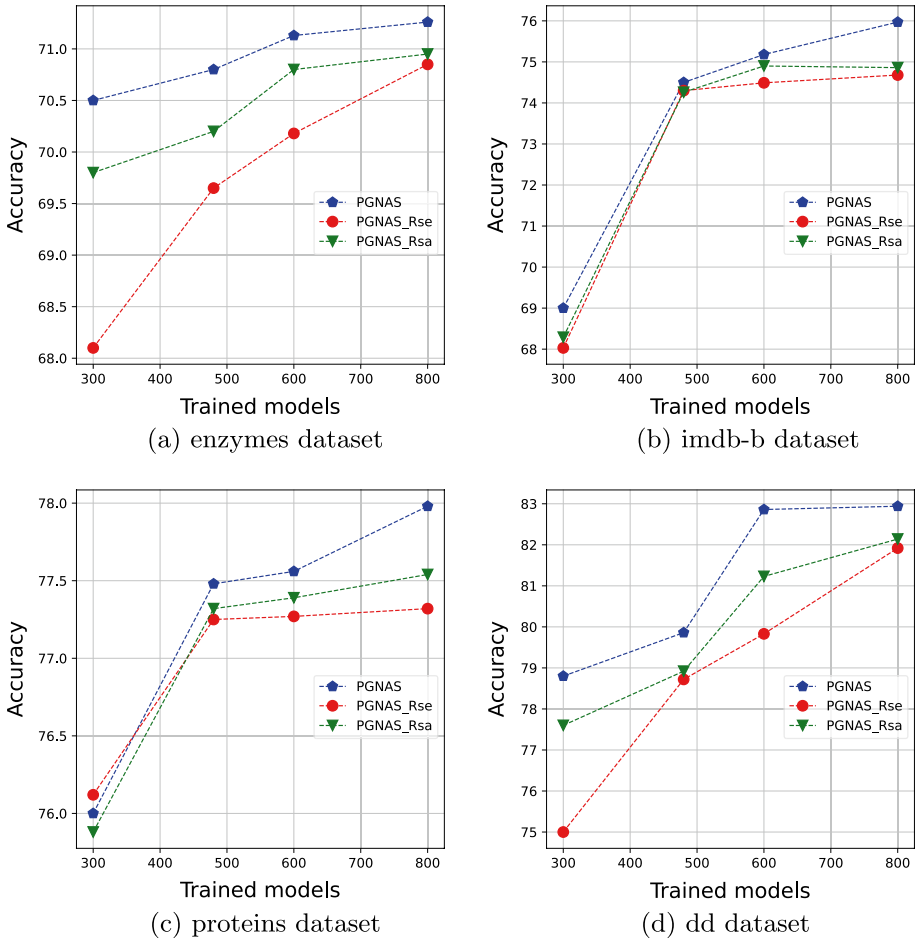


Fig. 4 Efficiency comparison

4.7 Statistical significance analysis

We make statistical tests and discuss the results of the statistical significance of the found differences.

Firstly, we use the Friedman test (Demsar, 2006) to verify the hypothesis of equal performance among compared methods. Before calculating the family-wise p-value, it is needed to reject the null hypothesis, which is then compared to methods that have equal performance. We apply non-parametric tests that are known to be suitable for comparing predictive models based on multiple data samples (Demsar, 2006; Garcia & Herrera, 2008). Compared to the parametric tests, the non-parametric tests require fewer assumptions (Demsar, 2006). In this experiment, we make the analysis with seven methods (PGNAS, BASGCN, PAS, CurGraph, PGCN, GraphNAS, and SNAG) and three datasets (the proteins dataset, imdb-b dataset, and dd dataset). We use the Friedman ranking test (García et al., 2010) to access the statistical significance of differences in the model's performance on multiple datasets, based on the average accuracy after repeated runs on

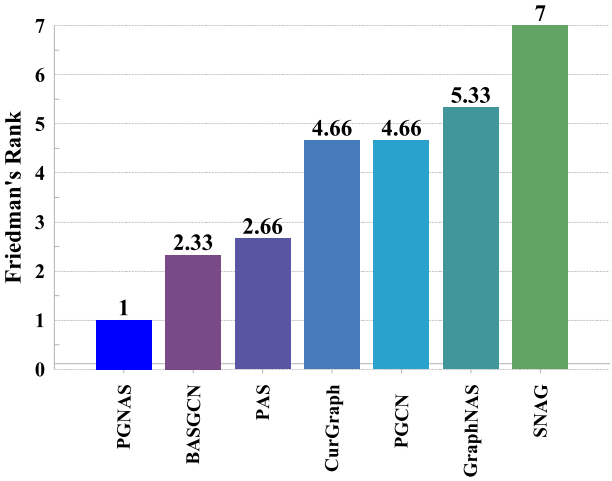


Fig. 5 Average Friedman ranking values of PGNAS and baseline methods. Average Friedman ranking values of PGNAS and baseline methods. PGNAS ranks first

each dataset. The overall performance ranking of compared methods to the accuracy metric is presented in Fig. 5. It can be seen that the proposed PGNAS ranks first position.

Then, we obtain family-wise p-values with Bonferroni correction. We calculate the chi-square as shown in Eq. 4,

$$\mathcal{X}_F^2 = \frac{12N}{k(k+1)}[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}] \tag{4}$$

where N is the number of datasets, k is the number of methods, and R_j the average ranking of the j^{th} method. With N=3 and k=7, we can get $\mathcal{X}_F^2 = 14.36$ Next, we calculate F_F as shown in Eq. 5.

$$F_F = \frac{(N-1)\mathcal{X}_F^2}{N(k-1) - \mathcal{X}_F^2} \tag{5}$$

In this experiment with seven algorithms and three datasets, $F_F = 7.8$. F_F is distributed according to the F-distribution with $7-1=6$ and $(7-1)\times(3-1)=12$ degrees of freedom. According to the F-distribution table, the critical value (CV) of F_F is 2.33 for $\alpha = 0.1$. As a result, the hypothesis of "equal" performance among compared methods is clearly rejected as $F_F > CV$.

Consequently, we use the Bonferroni correction (Holm, 1979) as a post-hoc test to compute family-wise p-values. In this computation, we set PGNAS as the control method and compute the family-wise p-values. The family-wise p-values between PGNAS and baseline methods are 0.0006672, 0.014019, 0.03763531, 0.03763531, 0.34470422, and 0.3447042 for SNAG, GraphNAS, CurGraph, PGCN, BASGCN, and PAS, respectively. It is revealed from the family-wise p-values that PGNAS shows significantly better performance over existing graph neural architecture search baseline frameworks with $\alpha = 0.1$.

Finally, we make the Nemenyi test (Demsar, 2006) to see how other baselines methods perform against each other. The Nemenyi test shows that every baseline method has significantly better performance over at most one other method. Figure 6 shows that every

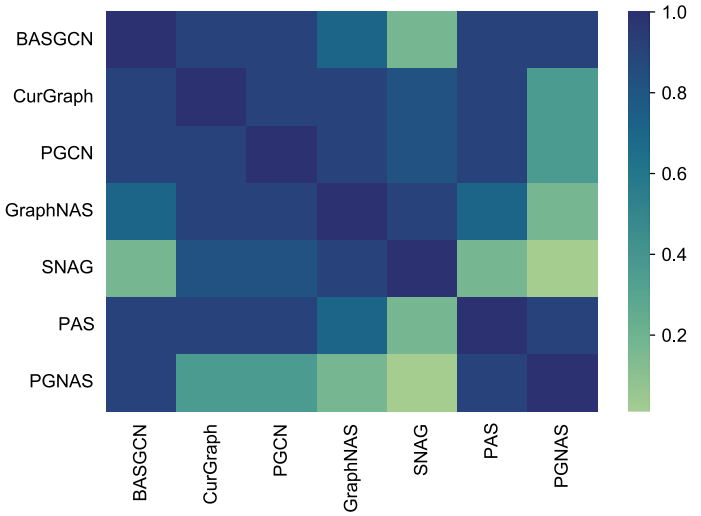


Fig. 6 Nemenyi test scores. Low score means significant difference between the performance of methods

baseline method has significantly better performance over at most one other method, which confirm the superiority of PGNAS among compared methods.

4.8 Ablation studies

4.8.1 Sampling method

To show that the sampling method is effective, we replace the sampling method with random sampling and note this framework as *PGNAS_RSa*. This means that we use random sampling instead of the controlled stratified random sampling described above. As shown in Fig. 4 PGNAS outperforms *PGNAS_RSa*. This result also indicates that GNN configurations sampled by *PGNAS_RSa* is not sufficiently representative of the global population and thus does not effectively represent the search space. Hence, the proposed sampling method is effective and should be used.

4.8.2 Choice of regression method

Here, we describe our results for conducting the final neural network performance. The objective of the performance distributions learner is to learn the validation accuracies distribution of the sampled GNN configurations and provide us with a curated list of promising GNN configurations for final validation. For this purpose, we try many regression models including *AdaBoostRegressor*, *RandomForestRegressor*, *MLPRegressor*, and *SGDRegressor*. For each experience, we train each regression model and evaluate their performances using three different evaluation metrics for comparison including R^2 *score*, spearman correlation (Myers & Sirois, 2003), and Kendall tau correlation (Ben Jemaa et al., 2015). To avoid reinforcing method bias toward these datasets, we use the bzr dataset in the experiment of the choice of the regression method. As seen in Table 4,

Table 4 Comparison of different regressor methods on the bzr dataset

Evaluation Metric	Regression model	Metric value
R^2 Score	MLPRegressor	0.81
	RandomForestRegressor	0.79
	SGDRegressor	0.74
	AdaBoostRegressor	0.23
Spearman correlation	MLPRegressor	0.86
	RandomForestRegressor	0.86
	SGDRegressor	0.69
	AdaBoostRegressor	0.09
Kendall tau correlation	MLPRegressor	0.76
	RandomForestRegressor	0.72
	SGDRegressor	0.63
	AdaBoostRegressor	0.07

MLPRegressor performs the best on most datasets, though not by a large margin. For the rest of this paper, we use *MLPRegressor* unless otherwise specified.

4.8.3 Performance distribution learner and encoding method

To prove the effectiveness of the encoding method, we replace the on-hot encoding with a binary encoding and compare the R^2errors on two datasets using the aforementioned performance distribution learner. As shown in Fig. 7, using the proposed one-hot encoding is more beneficial for encoded model representation learning. Meanwhile, this experience shows the power of the *MLPRegressor* over other models.

4.8.4 Influence of the search space

To show the importance of the search space we replace our proposed search space with the search space of GraphNAS and note this framework PGNAS_baseline. The GraphNAS search space does not include hyperparameters and contains the following functions.

- Aggregation function containing options such as mean, max, and sum.
- Multi-head attention containing options such as 1, 2, 4, 6, and 8.
- Hidden dimensions containing options such as 8, 16, 32, 64, 128, 256, and 512.
- Attention function containing options such as gat, gcn, cos, const,sym-gat, linear, and gene-linear.
- Activation function containing options such as tanh, sigmoid, relu, linear, relu6, elu, leaky_relu, and softplus.

As shown in Table 3, The result using our proposed search space is better than the result using the same search space as baselines and PGNAS still outperforms baseline models on all four datasets for both search spaces. This result demonstrates the effectiveness of PGNAS in searching efficiently neural architecture for graph classification and shows the importance of great search space in a graph neural architecture search framework. Indeed, With the proposed search space, we are able to obtain higher performance.

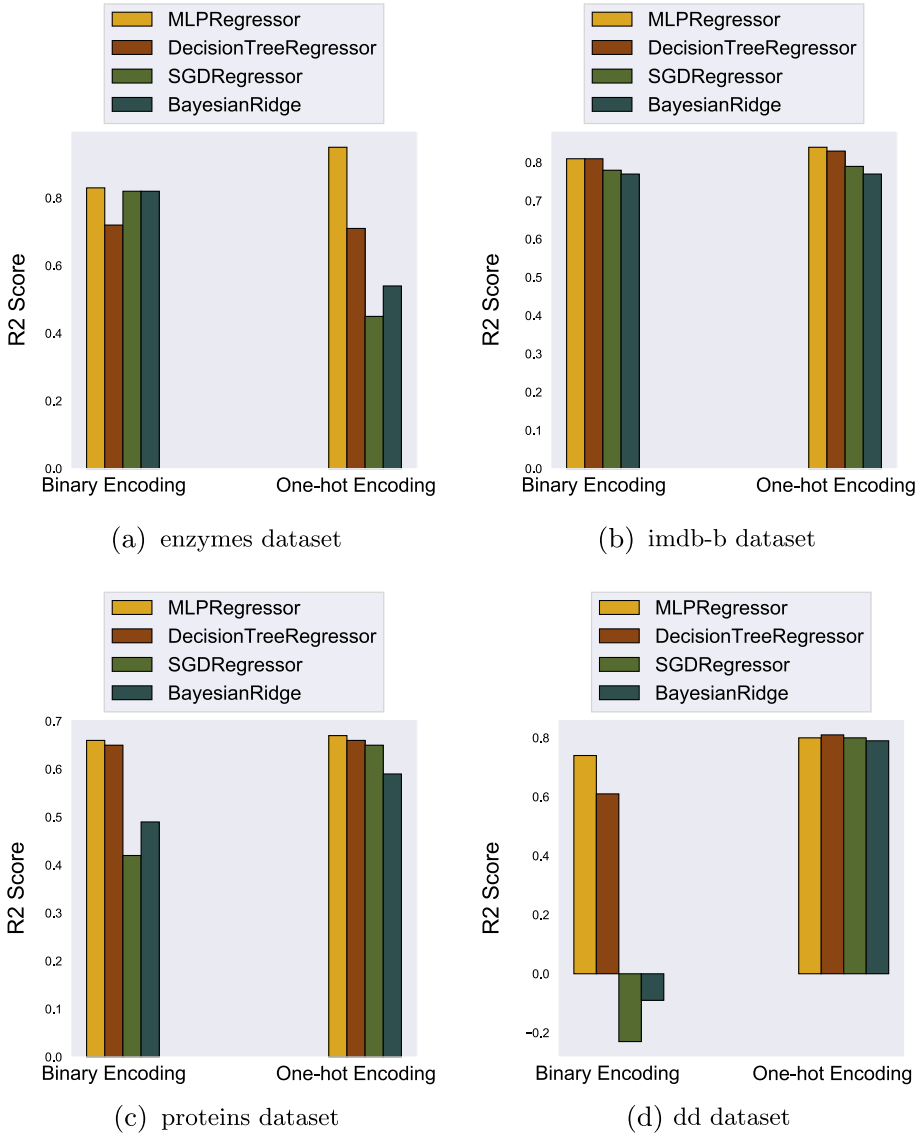


Fig. 7 R^2 score of different performance distribution learners using different GNN's configuration encoding methods

4.8.5 Robustness

To test the found differences between methods for statistical significance, we make repeated runs with different values for the random seed. As shown in Fig. 8, PGNAS is robust and can achieve great performance over repeated runs.

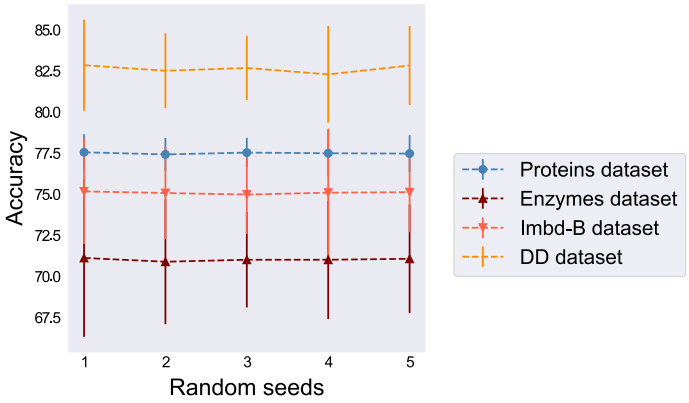


Fig. 8 PGNAS performance using different seed values on four benchmark datasets

5 Conclusion

In this work, we proposed an end-to-end framework, PGNAS, for the graph classification task. PGNAS is a performance predictor-based automated graph representation learning framework that predicts the performance of GNNs generated from the search space instead of training them. The performance prediction capability of PGNAS saves computation time. Moreover, PGNAS can broadly explore a search space with a cheap computation price, which enhances its scalability and efficiency. Besides, we use parallel computation to reduce the computation cost of the most expensive part of the framework, which is training a few GNN configurations to build the dataset used to train the predictor. Experiment results on the four benchmark datasets demonstrate that our proposed framework can obtain better performance than other manual and NAS GNN models.

Author contributions BMO: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing, review. JG: Conceptualization, Investigation, Project administration, Supervision, Validation, review. JC: Conceptualization, Data curation, Editing, Investigation,Resources, Validation, review. RA: Conceptualization, Investigation, Resources, Editing, review. TL: Conceptualization, Investigation, Validation, review.

Funding The work is supported by the National Natural Science Foundation of China (No. 61873288).

Data availibility statement All datasets used in the paper are publicly available.

Code availability Please email olouladem octard@csu.edu.cn to request code for the proposed method.

Declarations

Conflict of interest No conicts of interest or competing interests.

Ethics approval Not applicable

Consent to participate The manuscript is approved by all authors for publication.

Consent for publication All authors who participated in this study give the publisher permission to publish this work.

References

Bai, L., Cui, L., Jiao, Y., Rossi, L., & Hancock, E. (2020). Learning backtrackless aligned-spatial graph convolutional networks for graph classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 44,* 783–798.

Ben Jemaa, Z., Fournier-Prunaret, D., & Belghith, S. (2015). Kendall's tau based correlation analysis of chaotic sequences generated by piecewise linear maps. *In International Journal of Bifurcation and Chaos, 25*(13), 1550177.

Borgwardt, KM., Ong, C S, Schönauer, S, Vishwanathan, S. V. N., Smola, A J., & Kriegel, H-P. (2005). Protein function prediction via graph kernels. In *thirteenth international conference on intelligent systems for molecular biology*, pp. 47–56.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine, 34*(4), 18–42.

Cai, S, Li, L, Deng, J, Zhang, B, Zha, Zheng-J, Su, L, & Huang, Q. (2021). Rethinking graph neural architecture search from message-passing. In *IEEE conference on computer vision and pattern recognition*, CVPR 2021, virtual, June 19–25, 2021, pp. 6657–6666. Computer Vision Foundation / IEEE.

Chen, J, Gao, J, Chen, Y, Oloulade, M B, Lyu, T, & Li, Z. (2021). Graphpas: Parallel architecture search for graph neural networks. In *SIGIR '21: The 44th international ACM SIGIR conference on research and development in information retrieval*, Virtual Event, Canada, July 11–15, 2021, pp. 2182–2186. ACM.

Chen, J., Gao, J., Chen, Y., Moctard, O. B., Lyu, T., & Li, Z. (2022). Auto-GNAS: A parallel graph neural architecture search framework. *IEEE Transactions Parallel Distributed System, 33*(11), 3117–3128.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *In Journal of Machine Learning Research, 7,* 1–30.

Dobson, P. D., & Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *In Journal of molecular biology, 330*(4), 771–783.

Do, K., Tran, T., Nguyen, T., & Venkatesh, S. (2019). Attentional multilabel learning over graphs: A message passing approach. *Machine Learning, 108*(10), 1757–1781.

Fey, M, & Lenssen, J E. (2019). Fast graph representation learning with pytorch geometric. CoRR, arXiv: 1903.02428.

Gao, Y, Yang, H, Zhang, P, Zhou, C, & Hu, Y. (2020). Graph neural architecture search. In *proceedings of the twenty-ninth international joint conference on artificial intelligence*, pp. 1403–1409.

García, S., Fernández, A., Luengo, Ján., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *In Information sciences, 180*(10), 2044–2064.

Garcia, S., & Herrera, F. (2008). An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. *In Journal of Machine Learning Research, 9,* 1–18.

Guo, Zichao, Zhan, X, Mu, H, Heng, W, Liu, Z, Wei, Y, & Sun, J. (2020). Single path one-shot neural architecture search with uniform sampling. In computer vision - ECCV 2020 - 16th European conference, Glasgow, UK, August 23-28, 2020,In *Proceedings, Part XVI, volume 12361 of lecture notes in computer science*, pp. 544–560. Springer.

Hancock, J. T., & Khoshgoftaar, T. M. (2020). Survey on categorical data for neural networks. *Journal of Big Data, 7*(1), 28.

Holm, S. A. (1979). Simple sequentially rejective multiple test procedure. In Scandinavian Journal Of Statistics, pp.65–70.

Jain, B. J., & Wysotzki, F. (2004). Central clustering of attributed graphs. *Machine Learning, 56*(1–3), 169–207.

Kipf, T N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th international conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, conference track proceedings*. OpenReview.net.

Kolouri, S, Naderializadeh, N, Rohde, G K., & Hoffmann, H. (2021). Wasserstein embedding for graph learning. In *9th international conference on learning representations, ICLR 2021, Virtual Event, Austria*, May 3-7, 2021. OpenReview.net.

Li, Y, & King, I. (2020). Autograph: Automated graph neural network. In *neural information processing - 27th international conference, ICONIP 2020, Bangkok, Thailand, November 23-27, 2020, Proceedings, Part II, volume 12533 of lecture notes in computer science*, pp. 189–201. Springer.

Liang, J., Cui, J., Wang, J., & Wei, W. (2021). Graph-based semi-supervised learning via improving the quality of the graph dynamically. *Machine Learning, 110*(6), 1345–1388.

Liu, M, Gao, H, & Ji, S. (2020). Towards deeper graph neural networks. In *26th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 338–348.

Myers, L., & Sirois, Maria J. (2003). Spearman correlation coefficients, differences between. *In Encyclopedia of statistical sciences, 12*(1), 771–783.

Oloulade, B. M., Gao, J., Chen, J., Lyu, T., & Al-Sabri, R. (2021). Graph neural architecture search: A survey. *Tsinghua Science and Technology, 27*(4), 692–708.

Pasa, L., Navarin, Nò., & Sperduti, A. (2021). Polynomial-based graph convolutional neural networks for graph classification. *Machine Learning, 114*(4), 1205–1237.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., & Schomburg, D. (2004). BRENDA, the enzyme database: Updates and major new developments. *In Nucleic Acids Research, 32*(1–3), 431–433.

Search, Evolutionary Multi-objective Surrogate-Assisted Neural Architecture. (2020). Zhichao L, Kalyanmoy D, Erik D. G, W Banzhaf, and V N Boddeti. NSGANetV2. *In European computer vision conference, part,I*,pp. 35–51.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., & Borgwardt, K. M. (2011). Weisfeiler-Lehman graph kernels. *In Journal Machine Learning Research, 12*(1), 2539–2561.

Shi, R, Luo, J & Liu, Q. (2021). Fast evolutionary neural architecture search based on bayesian surrogate model. In *IEEE congress on evolutionary computation*, pp. 1217–1224.

Shi, H, Pi, R, Xu, H, Li, Z, Kwok, J T., & Zhang, T. (2020). Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In *annual conference on neural information processing systems*, pp 6–12.

Shi, M, Wilson, D A., Zhu, X, Huang, Y, Zhuang, Y, Liu, J , & Tang, Y. (2020). Evolutionary architecture search for graph neural networks. CoRR, arXiv:2009.10199.

Sourek, G., Zelezný, F., & Kuzelka, O. (2021). Beyond graph neural networks with lifted relational neural networks. *Machine Learning, 110*(7), 1695–1738.

Sutherland, J. J., O'Brien, L. A., & Weaver, D. F. (2003). Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships. *Journal of Chemistry Information Computer Science, 43*(6), 1906–1915.

Sutton, R S., McAllester, D A., Singh, S P., Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *advances in neural information processing systems*, pp. 1057–1063.

Uwents, W., Monfardini, G., Blockeel, H., Gori, M., & Scarselli, F. (2011). Neural networks for relational learning: An experimental comparison. *Machine Learning, 82*(3), 315–349.

Wang, Y , Wang, W, Liang, Y, Cai, Y, & Hooi, B. (2021). Curgraph: Curriculum learning for graph classification. In *WWW '21: The web conference 2021, Virtual Event* / Ljubljana, Slovenia, April 19–23, 2021, pp. 1238–1248. ACM / IW3C2.

Wei, L, Zhao, H, Yao, Q, & He, Z. (2021). Pooling architecture search for graph classification. In CIKM '21: The 30th ACM *international conference on information and knowledge management*, Virtual Event, Queensland, Australia, November 1 –5, 2021, pp. 2091–2100. ACM.

White, C, Neiswanger, W, & Savani, Y. (2021). BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *thirty-fifth AAAI conference on artificial intelligence*, pp. 10293–10301.

White, C, Zela, A, Ru, R, Liu, Y, & Hutter, F. (2021). How Powerful are performance predictors in neural architecture search?. In *annual conference on neural information processing systems*, pp. 28454–28469.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions Neural Networks Learning System, 32*(1), 4–24.

Yanardag, P, & Vishwanathan, S. V. N. (2015). Deep graph kernels. In *21th ACM SIGKDD international conference on knowledge discovery and data mining*. pp. 1365–1374.

Ying, Z , You, J, Morris, C, Ren, X, Hamilton, W L., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *advances in neural information processing systems 31: Annual conference on neural information processing systems 2018*, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pp. 4805–4815.

Ying, R, You, J, Morris, C, Ren, X, Hamilton, W L., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. arXiv preprint arXiv:1806.08804.

Yoon, M, Gervet, Théo, Hooi, B, & Faloutsos, C. (2020). Autonomous graph mining algorithm search with best speed/accuracy trade-off. In *20th IEEE international conference on data mining*, ICDM 2020, Sorrento, Italy, November 17–20, 2020, pp. 751–760. IEEE.

Yoshida, T., Takeuchi, I., & Karasuyama, M. (2021). Distance metric learning for graph structured data. *Machine Learning, 110*(7), 1765–1811.

You, J, Ying, Z, & Leskovec, J. (2020). Design space for graph neural networks. In *advances in neural information processing systems 33: Annual conference on neural information processing systems 2020*, NeurIPS 2020, December 6-12, 2020, virtual.

Zhang, Z, Wang, X, & Zhu, W. (2021). Automated machine learning on graphs: A survey. In *proceedings of the thirtieth international joint conference on artificial intelligence*, IJCAI 2021, Virtual Event / Montreal, Canada, 19–27 August 2021, pp. 4704–4712. ijcai.org.

Zhao, H, Wei, L, & Yao, Q. (2020). Simplifying architecture search for graph neural network. In *Proceedings of the CIKM 2020 Workshops co-located with 29th ACM international conference on information and knowledge management (CIKM 2020)*, Galway, Ireland, October 19-23, 2020, volume 2699 of CEUR Workshop Proceedings. CEUR-WS.org.

Zhao, H, Yao, Q, & Tu, W. (2021). Search to aggregate neighborhood for graph neural network. In *37th IEEE international conference on data engineering*, ICDE 2021, Chania, Greece, April 19-22, 2021, pp. 552–563. IEEE.

Zhou, K, Song, Q, Huang, X, & Hu, X. (2019). Auto-gnn: Neural architecture search of graph neural networks. CoRR, arXiv:1909.03184.

Authors and Affiliations

Babatounde Moctard Oloulade[1] · Jianliang Gao[1] · Jiamin Chen[1] · Raeed Al-Sabri[1] · Tengfei Lyu[1]

Babatounde Moctard Oloulade
olouladem octard@csu.edu.cn

Jiamin Chen
chenjiamin@csu.edu.cn

Raeed Al-Sabri
alsabriraeed@csu.edu.cn

Tengfei Lyu
tengfeilyu@csu.edu.cn

[1] School of Computer Science and Engineering, Central South University, Changsha 410083, China