Graph Masked Autoencoder Enhanced Predictor for Neural Architecture Search

Kun Jing, **Jungang Xu** and **Pengfei Li**
School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China
jingkun18@mails.ucas.ac.cn, xujg@ucas.ac.cn, lipengfei@ucas.ac.cn

Abstract

Performance estimation of neural architecture is a crucial component of neural architecture search (NAS). Meanwhile, neural predictor is a current mainstream performance estimation method. However, it is a challenging task to train the predictor with few architecture evaluations for efficient NAS. In this paper, we propose a graph masked autoencoder (GMAE) enhanced predictor, which can reduce the dependence on supervision data by self-supervised pre-training with untrained architectures. We compare our GMAE-enhanced predictor with existing predictors in different search spaces, and experimental results show that our predictor has high query utilization. Moreover, GMAE-enhanced predictor with different search strategies can discover competitive architectures in different search spaces. Code and supplementary materials are available here: https://github.com/kunjing96/GMAENAS.git.

1 Introduction

Neural architecture search (NAS) has drawn considerable attention, which can automate the process of designing neural architectures for a given task. Predictor-based NAS method has recently become popular, which roughly includes Gaussian processes [Swersky *et al.*, 2014; Kandasamy *et al.*, 2018; Ru *et al.*, 2020], neural networks [Ma *et al.*, 2019; Shi *et al.*, 2020; Wang *et al.*, 2019; White *et al.*, 2021a], tree-based methods [Luo *et al.*, 2020a; Siems *et al.*, 2020], and so on. The performance predictor for NAS performs better when more standard-trained architectures are available as training data. However, the training and evaluation of the architectures are extremely time-consuming, thus incurring a high initialization time.

The compromise of initialization time and quality of predictor is the key to predictor-based NAS. Some researchers try to reduce the average time cost of training through some techniques, e.g., parameter sharing [Pham *et al.*, 2018; Liu *et al.*, 2019] and proxy data [Park, 2019; Na *et al.*, 2021]. Other works attempt to train the predictor with few standard-trained architectures, e.g., semi-supervised learning [Luo *et al.*, 2020b; Tang *et al.*, 2020] and ensemble learning [Wu *et al.*, 2021; White *et al.*, 2021b]. We observe that the existing methods do not make full use of a large amount of available unlabeled data, i.e., all untrained architectures themselves in the search space.

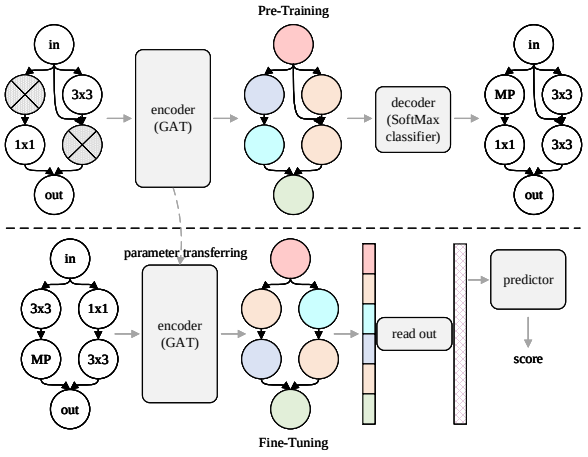


Figure 1: The overview of GMAE.

Based on the above observations, we propose a graph masked autoencoder (GMAE) that can help train an architecture performance predictor for neural architecture search, which can improve query utilization, as shown in Figure 1. Inspired by the success of self-supervised learning in natural language processing (NLP) and computer vision (CV), like BERT [Devlin *et al.*, 2019] and masked autoencoder (MAE) [He *et al.*, 2021], we utilize a large amount of unlabeled data (untrained architectures) in search space to pre-train the model. Specifically, we randomly mask some vertices in directed acyclic graphs (DAGs) that represent architectures and train the model to reconstruct the DAGs. After pre-training, we use only limited labeled data (the standard-trained architectures) to fine-tune the model, which is common practice when labeled data is insufficient. Furthermore, our pre-trained model can be transferred to different tasks in the same search space. The experimental results show that our pre-trained model can learn the encoding representation of neural architecture. Using limited standard-trained architectures, we can train the pre-trained model followed by a linear projection as a predictor whose generalization performance is improved.

We combine the predictor with different search strategies for NAS, and the architecture we discover has advanced performance on different NAS benchmarks. In order to verify the performance on the actual NAS task, we use 100 evaluated architecture samples on CIFAR-10 and train our predictor for NAS. The discovered architecture can achieve competitive results on CIFAR-10.

Our contributions are summarized as follows:

- We construct a neural architecture performance predictor through the self-supervised pre-training of GMAE, which can reduce the dependence on labeled data during the training of predictor and obtain higher generalization performance.
- We compare our predictor with others to prove that GMAE improves query utilization. We use different search algorithms combined with the fine-tuned predictor to prove the superiority of our method on different NAS benchmarks. On the actual NAS task, the architecture we discover achieved competitive results.
- We also performed ablation studies to demonstrate the effectiveness of our method. One main discovery is that generative self-supervised learning (e.g., masked graph reconstruction) is better than discriminant self-supervised learning such as graph contrastive learning in architecture performance estimation.

2 Related Work

Architecture performance predictor is first applied to NAS in PNAS [Liu *et al.*, 2018]. PNAS uses an ensemble of five LSTM-based predictors as the surrogate model that can read a serialized architecture to predict the validation accuracy. Representing an architecture as a computational graph, some works naturally try to predict the performance of architecture by graph neural networks (GNNs) [Wen *et al.*, 2020; Chen *et al.*, 2021a] and information propagation on graphs [Ning *et al.*, 2020]. For efficiency, the predictor should not have much predictive performance reduction when using fewer trained architectures as its training data. Some works [Wu *et al.*, 2021] propose ensemble models of many weaker predictors, which can achieve better predictive performance than any single model in the case of few architecture evaluations. The above method only uses supervised data to train predictors, while we utilize a large number of untrained architectures in the search space to help train models, which can reduce the dependence on supervised data.

There are two existing works closer to our work, which also use untrained architectures to help train predictors. One work is that a semi-supervised assessor [Tang *et al.*, 2020] predict their performances by a GNN, which employs an autoencoder to discover representations of architectures and construct a graph to capture the similarities of architectures. The difference is that we use a masked autoencoder instead of an autoencoder for better representation and we can independently predict the performance of one architecture without other architectures. Another work is that computation-aware neural architecture encoding (CATE) [Yan *et al.*, 2021] employs a pairwise pre-training scheme to learn computation-aware

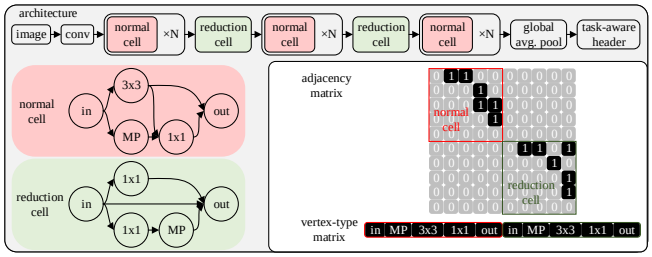


Figure 2: The architecture space and its representation.

encodings using Transformers with cross-attention. The encodings contain dense and contextualized computation information of neural architectures. Instead of using the data-hungry Transformer model, we use the GNN model. Although graph reconstruction pre-training is used, we take architectures as the input rather than computationally similar architecture pairs in the pre-training process, which can improve the efficiency of the pre-training.

3 Method

We describe the general form of architecture space in Section 3.1, where each architecture can be represented as a directed acyclic graph. We then introduce the encoder (Section 3.2) and decoder (Section 3.3) of our model. Finally, we introduce how to pre-train the model (Section 3.4), fine-tune the pre-trained model to build an architecture performance predictor (Section 3.5), and use the predictor for neural architecture search (Section 3.6).

3.1 Architecture Space

Following most NAS works [Pham *et al.*, 2018; Liu *et al.*, 2019], our architecture space is also based on cells or blocks. Each architecture consists of S stages, where each stage is composed of L-stacked normal cells followed by a reduction cell, except that the last stage is followed by a pooling layer and a task-aware header. For example, SoftMax classifier header is used in image classification.

Each cell is a DAG that consists of an ordered sequence of N vertices. In this work, we use an operation-on-vertex search space, i.e., each vertex represents a candidate operation and each edge represents the direction of the tensor (feature map) flow. The ingoing edge and outgoing edge represent the input and output feature map or tensor of the operation at the vertex, respectively. Therefore, each architecture can be represented by the DAGs corresponding to its stacked blocks or cells. For the convenience of representation, we directly merge the DAGs of normal and reduction cells into one DAG and use an adjacency matrix and a vertex-type matrix to represent the DAG, as shown in Figure 2.

3.2 Encoder

Our encoder is a graph attention network (GAT), which takes a DAG as its input, embeds candidate operations at all vertices in the DAG by an embedding layer, and then processes the embeddings using a series of GAT layers by message passing along the edges. Specifically, each K-head GAT

layer, parametrized by a weight $\omega^k \in \mathbb{R}^{d' \times d}$ and an attention mechanism parameter $a \in \mathbb{R}^{2d}$, can be formulated as

$$h'_i = \Big\Vert_{k=1}^{K} \sigma\Big(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij}^k \cdot \omega^k h_j\Big), \tag{1}$$

$$\alpha_{ij}^k = \frac{exp(LeakyReLU(a^\top[\omega^k h_i \| \omega^k h_j]))}{\sum_{k \in \mathcal{N}_i \cup \{i\}} exp(LeakyReLU(a^\top[\omega^k h_i \| \omega^k h_k]))}, \tag{2}$$

where $h_i \in \mathbb{R}^d$ and $h'_i \in \mathbb{R}^{d'}$ are the old and new features of vertex i, respectively; $\|$ represents concatenation; σ is the applied nonlinearity function by us; $\mathcal{N}_i$ indicates the neighbor of vertex i; α_{ij}^k is the normalized attention coefficient computed by the SoftMax function.

3.3 Decoder

Our decoder is a single linear projection followed by a SoftMax function, which decodes the vertex features provided by our encoder and predicts the candidate operation types at masked vertices of the DAG fed into our encoder. All vertices share the decoder parameters. The decoder is only used during the pre-training process to reconstruct the DAGs.

3.4 Pre-Training

Pre-training method. There are two types of mainstream self-supervised pre-trainings, including discriminant and generative methods. We choose the generative method, i.e., masked autoencoder task. Similar to all autoencoder, our graph masked autoencoder reconstructs the DAGs of architectures given the partial observations (the masked DAGs). Our GMAE pre-training is similar to masked language modeling [Devlin *et al.*, 2019] and masked autoencoder (MAE) [He *et al.*, 2021]. The difference is that the relationship between vertices is defined by the adjacency matrix of DAG.

Masking. We use the simple masking strategy to mask the vertice embeddings randomly without replacement, following a uniform distribution. Unlike BERT and MAE, the choice of masking ratios is sensitive to search space in our experiments.

Objective function. Our GMAE reconstructs the input DAGs by predicting the candidate operation types at masked vertices instead of all ones. Our decoder is a simple and single SoftMax classifier, whose output is the normalized probability of candidate operation types by the SoftMax function. We compute the cross-entropy error (CE) between the masked vertices of the original and reconstructed DAGs as the pre-training loss function, which can be formulated as

$$\mathcal{L}_{pt} = -\frac{1}{N_{\mathcal{MV}}} \sum_{i \in \mathcal{MV}} \sum_{c=1}^{\mathcal{C}} y_{ic} log(p_{ic}), \tag{3}$$

where $N_{\mathcal{MV}}$ is the number of masked vertices $\mathcal{MV}$ of the DAG $\mathcal{G}$; $\mathcal{C}$ represents the number of the candidate categories. If the real category of the operation at the vertex i is c, y_{ic} is 1; otherwise y_{ic} is 0. p_{ic} is the probability that the prediction category of the operation at the vertex i is c.

Evaluation. We evaluate our pre-trained model in two ways of supervised training: end-to-end and partial fine-tuning.

3.5 Fine-Tuning

In order to robustly evaluate our pre-training model, we need to repeatedly fine-tune the pre-trained model many times during evaluation and compute their average metrics. Fortunately, since we use very little supervised data for fine-tuning, the evaluation takes only several minutes.

Metrics. The Kendall rank correlation coefficient τ is a statistic to measure the rank correlation, i.e., the similarity of the orderings of the data, which ranges from 1 to -1. Due to the fact that the relative ranking of the architectures is more important than its specific performance value for NAS, we choose the Kendall ranking correlation coefficient τ as the metrics of our predictor, which is the same as most NAS works [White *et al.*, 2021b; Siems *et al.*, 2020].

End-to-end fine-tuning. We ignore our decoder and only use the encoder with the pre-trained parameters followed by the readout layer and the predictor (regressor) composed of several fully connected layers. Then, we fine-tune the new model with supervised data by end-to-end training.

Partial fine-tuning. We can also evaluate the pre-trained model by freezing the encoder parameters and updating only the predictor parameters. Therefore, partial fine-tuning usually takes less time than end-to-end fine-tuning.

Fine-tuning target. A simple idea is that mean squared error (MSE) between the predicted and real accuracies is used as the loss function. However, MSE loss forces the model to learn to predict the absolute performance rather than the relative ranking, which seems to be too strict to obtain the accurate predictions. Besides, lower MSE is sometimes not equivalent to better ranking. Inspired by learning to rank (LTR), the hinger loss and Bayesian personalized ranking (BPR) loss are commonly used for training the model that focuses only on relative ranking. We use BPR loss for fine-tuning our predictor, which can be formulated as

$$\mathcal{L}_{ft} = -\sum_{i,j \in S} log(sigmoid(s_i - s_j)), \tag{4}$$

where S is the pair-wise architecture set; s_i is the predicted performance of architecture i.

3.6 Search

After pre-training GMAE and fine-tuning our predictor, we use our predictor as evaluation strategy for NAS. In our experiments, we enhance four common search strategies, i.e., random search, reinforcement learning [Zoph and Le, 2017; Baker *et al.*, 2017], aging evolution [Real *et al.*, 2019], and Bayesian optimization [White *et al.*, 2021a], by our predictor. The algorithm descriptions are shown in Algorithm 1. The time complexity is $O(N_P \times N^2)$ where N is the maximum number of queries and N_P is the number of predictors in Bayesian optimization strategy ($N_P = 1$ in others).

4 Experiments

We first verify the effectiveness of the GMAE-enhanced predictor in different search spaces in Section 4.1. Furthermore, in Section 4.2, we demonstrate that the query utilization of

Algorithm 1 GMAE-Enhanced Predictor Based NAS (Different Search Strategies)

Draw N_0 architectures uniformly at random from search space A, train them, and add them into $history$.

Random Search Strategy
while $|history| < N - N_0$ **do**
  Train a predictor P on $history$.
  Draw C candidates uniformly at random from A.
  For each candidate $a \in C$, evaluate its score $P(a)$.
  Choose Top-K candidates by $P(a)$, train them, and add them into $history$.
end while

Reinforcement Learning Strategy
while $|history| < N - N_0$ **do**
  Train a predictor P on $history$.
  Select C candidates using policy π.
  For each candidate $a \in C$, evaluate its score $P(a)$.
  Choose Top-K candidates by $P(a)$, train them, and add them into $history$.
  Compute reward R, and update π by policy gradient.
end while

Aging Evolution Strategy
Append N_0 architectures into $population$.
while $|history| < N - N_0$ **do**
  Train a predictor P on $history$.
  Generate a set of C candidates by randomly mutating the M high-accuracy architectures selected by Tournament in $population$.
  For each candidate $a \in C$, evaluate its score $P(a)$.
  Choose Top-K candidates by $P(a)$, train them, and add them into $history$ and $population$.
  Remove the oldest individual from left of $population$.
end while

Bayesian Optimization Strategy
while $|history| < N - N_0$ **do**
  Train an ensemble P of predictors on $history$.
  Generate a set of C candidates by randomly mutating the Top-M high-accuracy architectures in $history$.
  For each candidate $a \in C$, evaluate its acquisition function $\phi(a, P)$.
  Choose Top-K candidates by $\phi(a, P)$, train them, and add them into $history$.
end while

return highest-accuracy model in $history$.

our search method surpasses other predictor-based methods. In addition to the experiments on two different NAS benchmarks, we also use our search method on the actual NAS task. Finally, in Section 4.3, we conduct some ablation studies.

NAS-Bench-101 The NAS-Bench-101 search space [Ying *et al.*, 2019] consists of 423K unique convolutional architectures. Each architecture is restricted to the cell space. The cell includes up to 7 vertices and at most 9 edges. The intermediate vertices can be either 1×1 convolution, 3×3 convolution, or 3×3 max pooling. The NAS-Bench-101 provides their validation and test accuracies on CIFAR-10.

NAS-Bench-301 (DARTS) The NAS-Bench-301 search space [Liu *et al.*, 2019; Siems *et al.*, 2020] is the larger cell-based space containing approximately 10^{18} architectures and their performances on CIFAR-10. Each normal or reduction cell is treated as a DAG with 7 vertices. Each directed edge represents one of the following operations: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, and skip connection.

Because NAS-Bench-301 is an operation-on-edge search space, we have to convert it into an operation-on-vertex space and then represent it in the way shown in Figure 2. Please see supplementary materials for details of converting.

4.1 Predictor Evaluation

To measure the quality of the pre-training model, we consider architecture performance prediction as its downstream task. Architecture performance prediction is the key to NAS. Predictors with high Kendall τ can often find better architectures combined with search strategies. After pre-training, we compare our GMAE-enhanced predictor with other popular architecture performance predictors [Ning *et al.*, 2020; Tang *et al.*, 2020; Xu *et al.*, 2021; Wen *et al.*, 2020] to prove that GMAE pre-training can help obtain a better architecture performance predictor. We directly use the reported results in their works for comparison. Our pre-training and fine-tuning setups are reported in the supplementary materials.

As shown in Figure 3, we report the Kendall τ values of different predictors with different numbers of training samples that are less than 5000 in NAS-Bench-101 space. When a large number of training samples are used for training, the performance of GMAE-enhanced predictor is only worse than GCN and ReNAS. This is because we use the same training hyperparameters for fine-tuning with different training samples and result in no convergence of the fine-tuning. However, for predictor-based NAS, because the architecture query takes a lot of time, we pay more attention to the performance of predictor when the number of training samples is small. When using less than 1500 training samples for predictor training, GMAE-enhanced predictor is superior to other predictors. In particular, when the number of training samples is less than 500, it is about 0.1 higher than the Kendall τ of the best predictor. The reason is that before the fine-tuning of our predictor, the pre-training of GMAE helps the encoder learn the better prior representation of the architecture. This prior representation is inherited by the predictor and helps the predictor perform better in the case of extremely few samples. The experimental results show that through pre-training, GMAE-enhanced predictor can obtain high Kendall τ under the condition of fewer training samples, i.e., it can help improve the predictor and make it better than most existing neural predictors.

Table 1 also reports the Kendall τ values of different numbers of training samples in NAS-Bench-301. As the number of samples increases, the growth of Kendall τ slows down and the Kendall τ values always maintain small standard deviations. GMAE-enhanced predictor can achieve Kendall τ close to 0.6 under the training of 100 training samples, which proves that it can work well in larger search spaces.

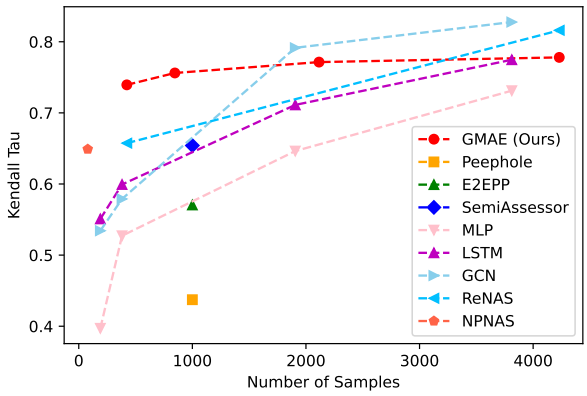


Figure 3: The comparison between GMAE-enhanced predictor and SOTA predictors with below 5000 training samples in NAS-Bench-101. It reports the mean and standard deviation of Kendall τ of 10 independent runs.

Samples	100	200	500	1000	10000
avg.	0.5988	0.6301	0.6530	0.6569	0.6809
std.	0.0175	0.0115	0.0162	0.0134	0.0090

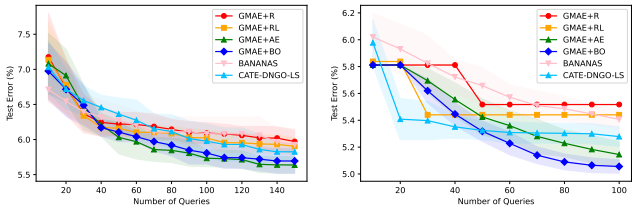
Table 1: The Kendall τ of GMAE with below 10000 training samples in NAS-Bench-301 (DARTS). It reports the mean and standard deviation of Kendall τ of 10 independent runs.

4.2 Search Evaluation

We use our predictor to enhance four predictor-independent search strategies, including random search (R), reinforcement learning (RL) [Zoph and Le, 2017; Baker *et al.*, 2017], aging evolution (AE) [Real *et al.*, 2019], and Bayesian optimization (BO) [White *et al.*, 2021a]. We compare our four NAS methods (GMAE-NAS) with two SOTA predictor-based NAS methods, i.e., BANANAS [White *et al.*, 2021a] and CATE [Yan *et al.*, 2021], to prove that our predictor can combine and enhance different search strategies to achieve higher query utilization [1] and discover better architectures. We replicate them using open-source code and default hyper-parameters provided in their works and report the test errors in the search process, as shown in Figure 4. Our search setup is reported in the supplementary materials.

The search process of the six NAS methods in NAS-Bench-101 space is shown in Figure 4(a). We report the test errors every 10 queries up to 150 queries. The random search and reinforcement learning strategies are close to BANANAS and worse than CATE. The reason is that random search and reinforcement learning strategies are the basic and outdated search strategies, which usually perform worse among the baseline search strategies. After being enhanced by our GMAE, their performance is close to the past SOTA BANANAS, which shows that GMAE is indeed effective. The Bayesian optimization and aging evolution strategies are significantly better than CATE. In particular, aging evolution strategy achieves a lower test error in NAS-Bench-101 and has higher query utilization. Compared with BANANAS and CATE, our method has a narrower error band, i.e., it is more stable. The experimental results demonstrate that our predictor with pre-training can improve the query utilization.

[1] Query utilization is defined as $\frac{N-N_Q}{N}$ where N_Q is the number of queries to discover the optimal architecture and N is the total number of architectures in the search space.



(a) NAS-Bench-101. (b) NAS-Bench-301.

Figure 4: The comparison between GMAE-enhanced strategies and SOTA NAS methods in NAS-Bench-101 (left) and NAS-Bench-301 (right). It reports the mean and standard deviation of test error of 10 independent runs. The maximum number of queries is set to 150 for NAS-Bench-101 and 100 for NAS-Bench-301.

Figure 4(b) also reports the test errors every 10 queries under the condition of up to 100 queries in NAS-Bench-301 space. The conclusion in NAS-Bench-301 is consistent with that in NAS-Bench-101 except for two differences: the gap between different methods is more significant; the Bayesian optimization strategy is superior to the aging evolution strategy in NAS-Bench-301 space. Experimental results prove that our method has more advantages in larger search spaces.

NAS-Bench-301 uses a surrogate model trained on 60k architectures to predict the performance of all the other architectures in the DARTS space. Therefore, the architecture performances can be inaccurate. Given that, we further prove the effectiveness of GMAE-NAS on the actual NAS task by training the queried architectures from scratch. Detailed setups of searching and training are reported in the supplementary materials. The average validation error of the last 5 epochs is computed as the performance label. For fair comparison, we compare our discovered architectures using the common evaluation script. Please see supplementary materials for the setup of architecture evaluation and our discovered architectures. Table 2 reports the test errors of different NAS algorithms. GMAE-NAS (BO) achieves competitive performance on the actual NAS task of DARTS space, which is superior to GMAE-NAS (AE) and other methods. In particular, GMAE-NAS (BO) has a 0.06 lower test error and is more stable than CATE-DNGO-LS that also use the idea of pre-training. This is consistent with the conclusion in Figure 4(b).

4.3 Ablation Study

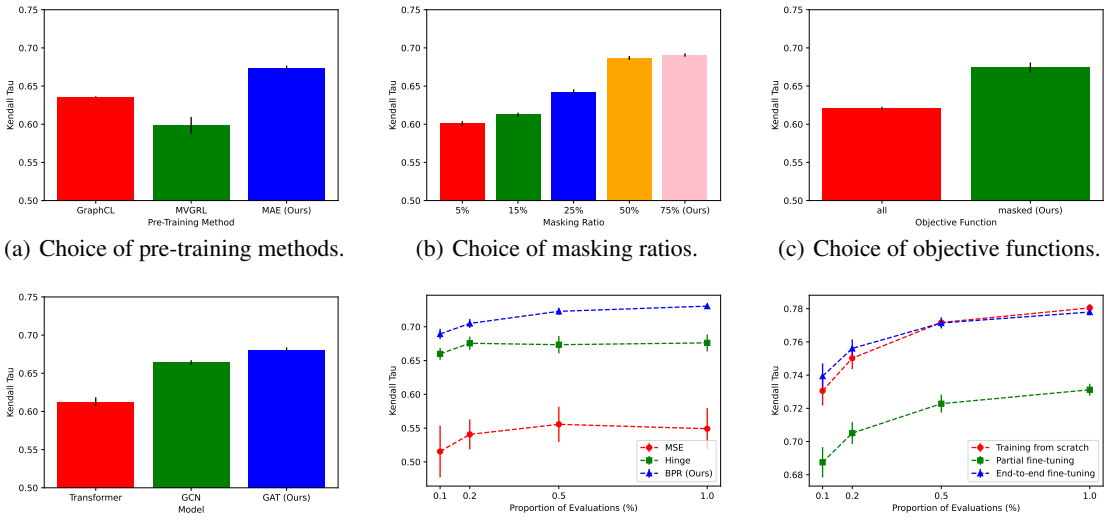
All ablation studies are performed on the NAS-Bench-101 benchmark. We report the Kendall τ values of partial fine-tuned predictors after pre-training.

Choice of Pre-Training Methods

As shown in Figure 5(a), compared with two types of graph contrastive learning, including GraphCL [You *et al.*, 2020] and MVGRL [Hassani and Ahmadi, 2020], our GMAE pre-training method can learn better graph representation that is more conducive to architecture performance prediction.

Algorithms	Test Error (%)	Parameters (M)	Search Cost (GPU days)	Search Type
CTNAS [Chen *et al.*, 2021b]	2.59±0.04	3.6	0.3	RL+neural comparator
RANK-NOSH [Wang *et al.*, 2021]	2.53±0.02	3.5	-	NOSH+neural predictor
BANANAS [White *et al.*, 2021a]	2.67±0.07	3.6	10.2	BO+neural predictor
CATE-DNGO-LS [Yan *et al.*, 2021]	2.56±0.08	2.9	3.3	BO+LS+neural predictor*
GMAE-NAS (AE)	2.56±0.04	4.0	3.3	AE+neural predictor*
GMAE-NAS (BO)	2.50±0.03	3.8	3.3	BO+neural predictor*

Table 2: The comparison of NAS algorithms in the NAS-Bench-301 (DARTS) search space. The search cost unit is GPU-days on a Tesla V100. * denotes that the predictor is enhanced by the pre-training. It reports the test error of 3 independent runs for our methods.



(a) Choice of pre-training methods. (b) Choice of masking ratios. (c) Choice of objective functions.

(d) Choice of models. (e) Choice of fine-tuning targets. (f) Choice of fine-tuning modes.

Figure 5: The ablation studies of pre-training methods, masking ratios, objective functions, models, fine-tuning targets, and fine-tuning modes, in NAS-Bench-101. It reports the mean and standard deviation of Kendall τ of 10 independent runs.

Choice of Masking Ratios

As shown in Figure 5(b), the masking ratio of 75% is the best option for NAS-Bench-101 space. But the masking ratio of 5% is the best for NAS-Bench-301 space that is larger and different from the NAS-Bench-101 space, which proves that the choice of masking ratios is sensitive to search space.

Choice of Objective Functions

Figure 5(c) shows that reconstructing the masked vertices is the better objective function than reconstructing all vertices.

Choice of Models

Figure 5(d) demonstrates that the data-hungry Transformer model fails to handle tasks with limited training data and GNN is the better choice. We choose the GAT model as a pre-training model and use it to build our predictor.

Choice of Fine-Tuning Targets

As shown in Figure 5(e), BPR loss is the optimal option across different numbers of training samples. Even in the case of limited training data, the model trained by BPR loss is extremely stable. That is because BPR loss pays more attention to ranking and it is smoother than Hinge loss.

Choice of Fine-Tuning Modes

Figure 5(f) shows that with extremely few training samples, our method can achieve a higher Kendall τ through end-to-end fine-tuning the pre-trained model, which proves that GMAE reduces the dependence on training data. However, when training with a large amount of data, our method does not perform well because the pre-training model does not have enough time to converge. Besides, it is worth noting that when we partial fine-tune our predictor, our predictor obtains higher Kendall τ than most existing predictors.

5 Conclusion

In this paper, we propose a GMAE that can help the training of an architecture performance predictor for NAS. It is demonstrated that through self-supervised pre-training, GMAE can reduce the dependence on training data to realize the predictor with high query utilization and improve the efficiency of predictor-based NAS. Our experimental results show its effectiveness and flexibility along with different search strategies in different search spaces. We anticipate that GMAE is a promising predictor-based NAS paradigm and it can bring vitality to the NAS community.

References

[Baker *et al.*, 2017] B. Baker, O. Gupta, N. Naik, et al. Designing neural network architectures using reinforcement learning. In *Proc. ICLR*, 2017.

[Chen *et al.*, 2021a] X. Chen, L. Xie, J. Wu, et al. Fitting the search space of weight-sharing NAS with graph convolutional networks. In *Proc. AAAI*, 2021.

[Chen *et al.*, 2021b] Y. Chen, Y. Guo, Q. Chen, et al. Contrastive neural architecture search with neural architecture comparators. In *Proc. CVPR*, 2021.

[Devlin *et al.*, 2019] J. Devlin, M. Chang, K. Lee, et al. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL-HLT*, 2019.

[Hassani and Ahmadi, 2020] K. Hassani and A. H. K. Ahmadi. Contrastive multi-view representation learning on graphs. In *Proc. ICML*, 2020.

[He *et al.*, 2021] K. He, X. Chen, S. Xie, et al. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

[Kandasamy *et al.*, 2018] K. Kandasamy, W. Neiswanger, J. Schneider, et al. Neural architecture search with bayesian optimisation and optimal transport. In *Proc. NeurIPS*, 2018.

[Liu *et al.*, 2018] C. Liu, B. Zoph, M. Neumann, et al. Progressive neural architecture search. In *Proc. ECCV*, 2018.

[Liu *et al.*, 2019] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. In *Proc. ICLR*, 2019.

[Luo *et al.*, 2020a] R. Luo, X. Tan, R. Wang, et al. Neural architecture search with GBDT. *arXiv preprint arXiv:2007.04785*, 2020.

[Luo *et al.*, 2020b] R. Luo, X. Tan, R. Wang, et al. Semi-supervised neural architecture search. In *Proc. NeurIPS*, 2020.

[Ma *et al.*, 2019] L. Ma, J. Cui, and B. Yang. Deep neural architecture search with deep graph bayesian optimization. In *Proc. WI*, 2019.

[Na *et al.*, 2021] B. Na, J. Mok, H. Choe, et al. Accelerating neural architecture search via proxy data. In *Proc. IJCAI*, 2021.

[Ning *et al.*, 2020] X. Ning, Y. Zheng, T. Zhao, et al. A generic graph-based neural architecture encoding scheme for predictor-based NAS. In *Proc. ECCV*, 2020.

[Park, 2019] M. Park. Data proxy generation for fast and efficient neural architecture search. *arXiv preprint arXiv:1911.09322*, 2019.

[Pham *et al.*, 2018] H. Pham, M. Y. Guan, B. Zoph, et al. Efficient neural architecture search via parameter sharing. In *Proc. ICML*, 2018.

[Real *et al.*, 2019] E. Real, A. Aggarwal, Y. Huang, et al. Regularized evolution for image classifier architecture search. In *Proc. AAAI*, 2019.

[Ru *et al.*, 2020] B. X. Ru, X. Wan, X. Dong, et al. Neural architecture search using bayesian optimisation with weisfeiler-lehman kernel. *arXiv preprint arXiv:2006.07556*, 2020.

[Shi *et al.*, 2020] H. Shi, R. Pi, H. Xu, et al. Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In *Proc. NeurIPS*, 2020.

[Siems *et al.*, 2020] J. Siems, L. Zimmer, A. Zela, et al. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

[Swersky *et al.*, 2014] K. Swersky, D. Duvenaud, J. Snoek, et al. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.

[Tang *et al.*, 2020] Y. Tang, Y. Wang, Y. Xu, et al. A semi-supervised assessor of neural architectures. In *Proc. CVPR*, 2020.

[Wang *et al.*, 2019] L. Wang, Y. Zhao, Y. Jinnai, et al. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019.

[Wang *et al.*, 2021] R. Wang, X. Chen, M. Cheng, et al. RANK-NOSH: efficient predictor-based architecture search via non-uniform successive halving. *arXiv preprint arXiv:2108.08019*, 2021.

[Wen *et al.*, 2020] W. Wen, H. Liu, Y. Chen, et al. Neural predictor for neural architecture search. In *Proc. ECCV*, 2020.

[White *et al.*, 2021a] C. White, W. Neiswanger, and Y. Savani. BANANAS: bayesian optimization with neural architectures for neural architecture search. In *Proc. AAAI*, 2021.

[White *et al.*, 2021b] C. White, A. Zela, B. Ru, et al. How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*, 2021.

[Wu *et al.*, 2021] J. Wu, X. Dai, D. Chen, et al. Weak NAS predictors are all you need. *arXiv preprint arXiv:2102.10490*, 2021.

[Xu *et al.*, 2021] Y. Xu, Y. Wang, K. Han, et al. Renas: Relativistic evaluation of neural architecture search. In *Proc. CVPR*, 2021.

[Yan *et al.*, 2021] S. Yan, K. Song, F. Liu, et al. CATE: computation-aware neural architecture encoding with transformers. In *Proc. ICML*, 2021.

[Ying *et al.*, 2019] C. Ying, A. Klein, E. Christiansen, et al. Nas-bench-101: Towards reproducible neural architecture search. In *Proc. ICML*, 2019.

[You *et al.*, 2020] Y. You, T. Chen, Y. Sui, et al. Graph contrastive learning with augmentations. In *Proc. NeurIPS*, 2020.

[Zoph and Le, 2017] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proc. ICLR*, 2017.

A Conversion to Operation-On-Vertex Search Space

As shown in Figure 6, we convert operation-on-edge architecture to operation-on-vertex architecture via a simple conversion. Following the description in Section 3.1, an architecture can also be represented by an adjacency matrix and a vertex-type matrix.

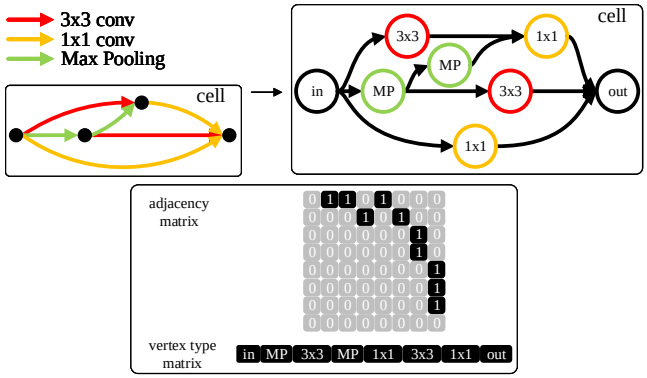


Figure 6: The conversion and representation of operation-on-vertex search space.

B Experimental Setups

We randomly sample the architectures from the whole search space for pre-training. For training and fine-tuning, we use a specified number of training samples as the training set and randomly sample 10000 architectures as the validation set. During training and fine-tuning in NAS-Bench-301, we pre-sample 100000 architectures as the sampling space. Other training setups are as follows.

Pre-training. The setups of pre-training in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces are demonstrated as in Table 3.

Hyper-parameters	NAS-Bench-101	NAS-Bench-301 (DARTS)
batch size	2048	4096
iterations	10000	10000
lr	0.0001	0.001
lr schedule	cosine	cosine
weight decay	0.0	0.0
dropout	0.0	0.2
masking ratio	0.75	0.05
optimizer	AdamW	AdamW
layers	16	12
model dims	32	32
activation	prelu	prelu
base model	GAT	GAT

Table 3: The setups of pre-training in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces.

Training from scratch. The setups of training from scratch in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces are demonstrated as in Table 4.

Hyper-parameters	NAS-Bench-101	NAS-Bench-301 (DARTS)
epochs	75	75
steps pre epoch	10	10
criterion	bpr	bpr
lr (encoder)	0.01	0.01
lr (predictor)	0.01	0.01
lr schedule	cosine	cosine
weight decay	0.0001	0.001
child-tuning reserve	0.5	0.8
child-tuning mode	F	D
dropout (encoder)	0.2	0.1
dropout (predictor)	0.2	0.1
optimizer	AdamW	AdamW
layers	16	12
model dims	32	32
activation	prelu	prelu
base model	GAT	GAT

Table 4: The setups of training from scratch and partial fine-tuning in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces.

Partial fine-tuning. The setups of partial fine-tuning NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces are the same as the setups of training from scratch, as shown in Table 4. But we freeze the parameters of the encoder.

End-to-end fine-tuning. The setups of end-to-end fine-tuning in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces are demonstrated as in Table 5.

Hyper-parameters	NAS-Bench-101	NAS-Bench-301 (DARTS)
epochs	100	25
steps pre epoch	10	10
criterion	bpr	bpr
lr (encoder)	0.001	0.001
lr (predictor)	0.01	0.001
lr schedule	cosine	cosine
weight decay	0.001	0.01
child-tuning reserve	0.5	1.0
child-tuning mode	D	D
dropout (encoder)	0.1	0.0
dropout (predictor)	0.1	0.5
optimizer	AdamW	AdamW
layers	16	12
model dims	32	32
activation	prelu	prelu
base model	GAT	GAT

Table 5: The setups of end-to-end fine-tuning in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces.

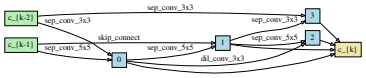
Search. The setups of search in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces are demonstrated as in Table 6.

Hyper-parameters	NAS-Bench-101	NAS-Bench-301 (DARTS)
N	150	100
N_0	20	20
K	10	10
C	100	100
M	1	1
ϕ	ITS	ITS

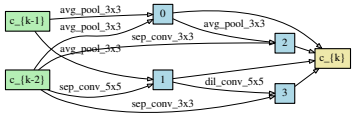
Table 6: The setups of search in NAS-Bench-101 and NAS-Bench-301 (DARTS) search spaces. ITS denotes Independent Thompson sampling (ITS) acquisition function.

C Best Discovered Architecture on the Actual NAS Task

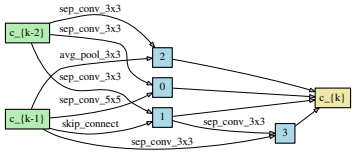
Figure 7 shows our best discovered architecture on the actual NAS task.



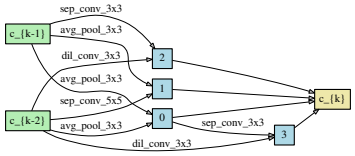
(a) Normal cell of GMAE-NAS (AE).



(b) Reduction cell of GMAE-NAS (AE).



(c) Normal cell of GMAE-NAS (BO).



(d) Reduction cell of GMAE-NAS (BO).

Figure 7: The best discovered architecture on the actual NAS task.

D Training the Discovered Architecture on CIFAR-10

We train the quired architecture and the discovered architectures on the CIFAR-10 dataset strictly following the hyperparameter setup of DARTS [Liu *et al.*, 2019]. The setups are shown in Table 7.

Hyper-parameters	Search	Evaluation
batch size	96	96
lr	0.025	0.025
momentum	0.9	0.9
weight decay	0.0003	0.0003
epochs	50	600
init channels	32	36
layers	8	20
auxiliary	0.4	0.4
cutout	16	16
drop path prob	0.2	0.2
grad clip	5.0	5.0

Table 7: The setups of training in search and evaluation process.