A Neural Architecture Predictor based on GNN-Enhanced Transformer

Xunzhi Xiang **Kun Jing** **Jungang Xu**
School of Computer Science and Technology, University of Chinese Academy of Sciences, China
xiangxunzhi21@mails.ucas.ac.cn

Abstract

Neural architecture performance predictor is an efficient approach for architecture estimation in Neural Architecture Search (NAS). However, existing predictors based on Graph Neural Networks (GNNs) are deficient in modeling long-range interactions between operation nodes and prone to the problem of over-smoothing. Furthermore, some Transformer-based predictors use simple position encodings to improve performance via self-attention mechanism, but they fail to fully exploit the subgraph structure information of the graph. To solve this problem, we propose a novel method to enhance the graph representation of neural architectures by combining GNNs and Transformer blocks. We evaluate the effectiveness of our predictor on NAS-Bench-101 and NAS-bench-201 benchmarks, the discovered architecture on DARTS search space achieves an accuracy of 97.61% on CIFAR-10 dataset, which outperforms traditional position encoding methods such as adjacency and Laplacian matrices. The code of our work is available at https://github.com/GNET.

1 INTRODUCTION

Neural Architecture Search (NAS) has recently attracted considerable attention for automatically designing neural architectures (Liu et al., 2019; Fang et al., 2020). Early researchers evaluate the neural architecture through standard training. Due to the low cost, the predictor-based NAS methods (White et al., 2021; Luo et al., 2018, 2020; Li et al., 2020b; Ning et al., 2020) have become popular. Performance predictor evaluation strategy is implemented through neural networks, mapping the architecture space to absolute or relative performance values. How to encode discrete architectures into continuous feature representations determines the performance ceiling of the predictor.

The neural architecture performance predictor is first applied to PNAS (Liu et al., 2018). Recent works focus on using different coding methods to represent neural architectures. Some methods represent a neural architecture as a computational graph, predicting architecture performance through Graph Neural Networks (GNNs) (Chen et al., 2021; Wen et al., 2020; Li et al., 2020a; Shi et al., 2020) . Recently, Transformer-based models (Vaswani et al., 2017; Devlin et al., 2019) that have proved to be successful in Natural Language Processing (NLP) offer the potential to further improve prediction performance. TNASP (Lu et al., 2021) adopts Transformer to encode neural network for the first time and achieves better performance compared to some existing methods. However, both GNN-based and Transformer-based predictors have shortcomings. With many different message-passing strategies (Kipf and Welling, 2017; Velickovic et al., 2018; Hamilton et al., 2017) having been proposed, some critical limitations are uncovered in this class of GNNs. For example, Graph neural networks are prone to over-smoothing (Li et al., 2018; Chen et al., 2020; Oono and Suzuki, 2020) or oversquashing (Alon and Yahav, 2021) as the number of layers deepens and lack the ability to model over long distances. In the meantime, simple connection embeddings of the adjacency matrix in the Transformer-based predictor do not fully utilize the structural information of the neural architecture.

Inspired by GNN-Transformer hybrid networks (Rampásek et al., 2022; Mialon et al., 2021; Dwivedi and Bresson, 2020; Rong et al., 2020), we propose a novel architecture encoding method: A Neural Architecture Predictor based on GNN-Enhanced Transformer

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

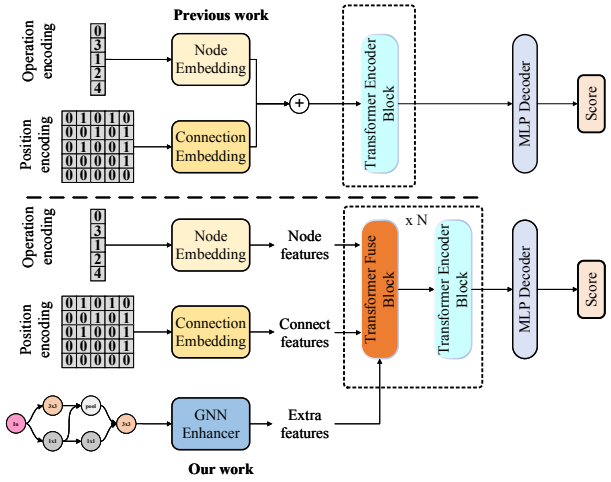


Figure 1: Previous Transformer-based NAS predictor and GNET.

(GNET). As shown in Figure 1, previous works use node embedding and connection embedding to transform node operations and adjacency matrices to obtain node and connection features respectively and then follow the approach of transformer to add the two features to form a token for calculation. Different from them, we propose a feature fusion block based on this approach, which uses GNN to generate additional architectural information instead of simple matrix coding to enhance graph representation. The improvements of our work mainly includes two parts: one is the design of feature fusion methods, and the other is the design of feature selection methods. We adopt two feature fusion methods, one is based on Cross-Attention, another is based on Structure-Aware. The Cross-Attention based method is generally used for feature fusion between different modalities. We believe that GNN has strong enough ability to represent graph nodes and connections, which can be served as a separate modal sequence for information exchange with Self-Attention sequences. The Structure-Aware based method is first applied in SAT (Chen et al., 2022), which is mainly used to solve the problem that most current methods only encode the positional relationships between nodes without explicitly encoding the structural relationships. The same convolution or pooling operation may play a significant role in different subgraph structures. There are also two methods for feature selection. One method is to input node features as a separate feature sequence for feature fusion; Another method is to input connection features as a sequence of individual features into fuse block for feature fusion. After experiments, the method of using Cross-Attention to fuse connection features and GNN features achieves the best results, which outperforms the current state-of-the-art method.

We verify the performance of our predictor on NAS-bench-101 (Ying et al., 2019) and NAS-bench-201 (Dong and Yang, 2020), using Kendall's tau as the performance testing standard. The experimental results demonstrate that it is feasible and effective to enhance neural architecture representations by using GNNs, resulting in a stronger predictor. Besides, we combine the predictor with evolutionary algorithms and conduct a real architecture search on the DARTS search space, the discovered architecture can achieve competitive results on CIFAR-10.

Our contributions are summarized as follows:

- We propose a method to combine GNN with Transformer network and design a feature fusion module to produce a more powerful neural architecture predictor.
- We propose two methods for feature fusion: Cross-Attention-based method and Structure-Aware-based methods and two methods for feature selection: Node-based method and Connection-based method.
- We compare our predictor with others though experiments, and experimental resutls demonstrate the superiority of our approach on different NAS benchmarks.

2 RELATED WORK

2.1 GNN-based Neural Architecture Predictors

Most of works regard the network architecture as a stack of two repeated sub-architectures (Normal cell and Reduction cell) and each cell is a DAG that consists of an ordered sequence of N vertices. Neural Predictor (Wen et al., 2020) encodes each node in the architecture using the average of two modified GCN layers. GMAE (Jing et al., 2022) uses graph reconstruction as the pretext task to learn the Architecture Representation Information. DCLP (Zheng et al., 2023) combines GNN and contrastive learning with curriculum learning guidance to explore the full use of the information contained in unlabeled data.

Although each of these methods has made improvements, they are still limited by the performance ceiling of GNN. For example, the computational cost in these methods increases quadratically with the size of the adjacency matrix. And due to the lack of long-distance modeling ability, GNN is overly dependent on increasing the number of layers to enhance graph information interaction. Therefore, in our proposed

method, we combine Self-Attention module to make up this deficiency.

2.2 Transformer-based Neural Architecture Predictors

The Transformer-based model, initially introduced in the field of Natural Language Processing (NLP), is designed to capture global dependencies and enable parallel computation through the implementation of the Multi-Head Self-Attention (MHSA) mechanism. For modeling neural networks, CATE (Yan et al., 2021) employs a pairwise pre-training scheme to learn computation-aware encodings using Transformers with cross-attention. TNASP performs feature embedding for adjacency matrix and node operation respectively, and then concatenates them together as the input of Transformer block.

We believe that embedding the Laplacian matrix simply cannot make full use of the structural information of neural architecture. In our method, we uses GNNs to extract additional features, and fuses this features with Transformer sequences to enhance the representation of neural architectures. Experimental results demonstrate the superiority of our method.

3 METHOD

3.1 Formal Problem Definition

The primary objective of Neural Architecture Search (NAS) is to identify an optimal neural network architecture tailored for a specific task. This process initiates with a predefined set of operations, from which a plethora of candidate architectures are generated via a systematic search strategy. Subsequently, these candidate architectures undergo training and validation processes to ascertain their performance metrics. However, the necessity to train each potential architecture from the ground up presents a substantial demand for time and computational resources. In this context, developing efficient and precise evaluation strategies is crucial for enhancing the expediency of NAS algorithms. Within this framework, predictor-based NAS methods emerge as a pivotal innovation. These methods employ performance predictors, which are designed to swiftly estimate the accuracy of various architectures, thereby obviating the need for exhaustive training of each architecture to determine its performance.

Previous work (Lu et al., 2021) proposes the first application of Transformer to encode discrete neural architectures into continuous feature representations, which can be formulated as:

$$e = \text{Transformer}(A, k) \tag{1}$$

where $A \in \mathbb{R}^{N\times N}$ denotes the adjacency matrix, representing the directed acyclic connections between nodes. N denotes the number of the nodes. $k \in \mathbb{R}^{N\times C}$ denotes the feature matrix, encapsulating the characteristics of the nodes, C denotes the output dimension of the embedding layer. Our method is based on this basis and combines Transformer and GNN to obtain better results.

3.2 Overall Framework

Figure 2 shows the overall framework of our proposed method called GNET. Firstly, we encode the discrete architecture into continuous feature vectors: node embedding and connection embedding are used to encode the node operations and node connection of the architecture into node features and connection features, respectively. Unlike TNASP, we do not simply concatenates two vectors but input them into the fuse block for specific feature fusion. Afterward, we use GNN to encode the directed acyclic graph corresponding to the architecture and extract additional features for fusion with previous features. Finally, we use a simple decoder composed of fully connected layers and activation layers to obtain the final architecture score. Due to the page size limitation, Figure 2 shows only the best one of our methods, the rest will be given in the supplementary material.

3.3 GNN Predictor Enhancer

To enhance the expression of neural architecture in Transformer, we use GNN as a predictor enhancer to introduce additional information and feed it into the fusion block. GNN takes a DAG as input, embeds candidate operations at all vertices in the DAG with an embedding layer, and then processes the embeddings using a series of GNN layers by message passing along the edges.

We try different graph neural networks and find GAT is the optimal one through experiments. For each K-head GAT layer, it can be formulated as follows:

$$h_i' = \Big\Vert_{k-1}^{K} \sigma\Big(\sum_{j\in N_i\cup\{i\}} \alpha_{ij}^k \cdot \omega^k h_j\Big) \tag{2}$$

$$a_{ij}^k = \frac{\exp(\text{LReLU}(a^T[\omega^k h_i \| \omega^k h_j]))}{\sum_{k\in N_i\cup\{i\}} \exp(\text{LReLU}(a^T[\omega^k h_i \| \omega^k h_k]))} \tag{3}$$

where h_i' and h_i are the old and new features of node v_i, $\|$ denotes concatenation, α_{ij}^k are normalized atten-

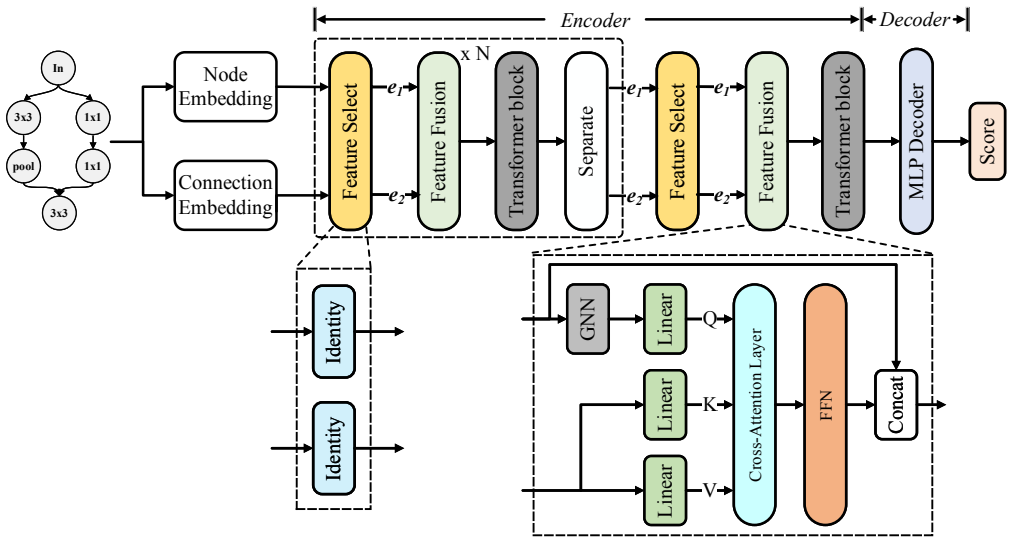


Figure 2: The overall framework of GNET. GNET mainly consists of an encoder and a decoder. The encoder is mainly composed of Feature Select block, Feature Fusion block and Transformer block. The decoder is a single linear projection followed by a SoftMax function.

tion coefficients computed by the k-th attention mechanism and $\cdot^T$ denotes transposition, ω^k is the corresponding input linear transformation's weight matrix, N_i denotes the neighbor of node v_i.

3.4 Feature-fusion Method

One of the two feature fusion methods in the fuse block we propose is based on Cross-Attention. The attention mechanism can be abstractly summarized as:

$$\text{Attention}(q, k, v) = f_{sim}(q, k)v \tag{4}$$

where $f_{sim}(q, k)v$ is a function used to compute the similarity scores between queries(q) and keys(k). The output of the attention mechanism is the weighted sum of values(v) based on similarity scores.

In our model, both Multi-head Self-Attention (MSA) and Multi-head Cross-Attention (MCA) are used, where MSA is adopted in the encoder after MCA to model the intra-relationship of operations sequence. MCA is used to fuse extra structure information generated by GNN and node representation information generated by MSA. The above two attention modules use $Softmax()$ as the similarity scoring function, which can be formulated as:

$$\text{CR}(f_1, f_2) = \delta(\frac{\phi_q(\psi(f_1, G))\phi_k(f_2)^T}{\sqrt{d_k}})\phi_v(f_2) \tag{5}$$

$$\psi(x, G) = \text{GNN}_G^l(x) \tag{6}$$

where f_1 and f_2 are the inputs of the fuse block, $\phi_q, \phi_k, \phi_v \in \mathbb{R}^{C\times C}$ are learnt parameter matrics, GNN_G^l denotes an arbitrary GNN model with l layers applied to graph G with node features, δ is Softmax activation function.

Another method is based on Structure-Aware. Self-Attention in the Transformer can only capture the attributed similarity between a pair of nodes. The problem of this method is that it cannot distinguish nodes that have different substructures but share identical operational characteristics. In order to also incorporate the structural similarity between nodes, we adopt a more generalized calculation method that additionally consider the local substructures around each node. A straightforward way to extract local structural information at node u is to apply any existing GNN model to the input graph with node features X and take the output as the subgraph representation of node u. This process can be formulated as:

$$\text{ST}(f_1, f_2) = \delta(\frac{\phi_q(\psi(f_1, G))\phi_k(\psi(f_2, G)^T}{\sqrt{d_k}})\phi_v(f_2) \tag{7}$$

$$\psi(x, G) = \text{GNN}_G^l(x) \tag{8}$$

where f_1 and f_2 are the inputs of the fuse block, $\phi_q, \phi_k, \phi_v \in \mathbb{R}^{C\times C}$ are learnt parameter matrics, GNN_G^l denotes an arbitrary GNN model with l layers applied to graph G with node features.

3.5 Node-based Fusion Transformer Predictor

Our Neural Architecture Predictor model is built upon the Transformer encoder architecture that consists of a semantic embedding layer, N Transformer blocks and N Fuse blocks stacked on top. Given a DAG G as the input, we first get the operation feature $e_1 \in \mathbb{R}^{N\times C}$ by transforming the operation vector k with an embedding matrix $E \in \mathbb{R}^{N\times C}$.

$$e_1^0 = E(k) \tag{9}$$

In this method, we choose e_1 as the input of fuse-block. After the node features enter the fuse block, the output obtained by fusing with the additional architecture information generated by GNN is used as the input of the next layer of Self-Attention. Finally, after processing through Self-Attention, the output of MSA is used as the input of the subsequent fuse-block. This process can be described as:

$$f_{fuse}^k = \text{Fuse-Block}(e_1^k, e_1^k) \tag{10}$$

$$f_{self}^k = \text{Self-Attention}(f_{fuse}^k) \tag{11}$$

$$e_1^{k+1} = f_{self}^k \tag{12}$$

where k denotes the current number of layers, Fuse-Block denotes one of the two fusion methods introduced in Section 3.3.

Finally, we choose a simple linear projection as the decoder, which decodes the vertex features provided by Transformer and gets the performance score.

$$score = \text{MLP}(F) \tag{13}$$

where F denotes the output of the encoder.

3.6 Connection-based Fusion Transformer Predictor

To further improve our method, we argue if we concatenate the connection features into node features and map the given structure into a single sequence as the input of Self-Attention, the ability to handle sequences in variable length and to model long-range dependency will be fully utilized. Similar to the method introduced in Section 3.4, we add a connection embedding layer which maps the Laplacian matrix (L) to a continuous position feature vector $e_2 \in \mathbb{R}^{N\times C}$. The connection embedding layer consists of a simple MLP.

In this method, we choose e_1 and e_2 as the input of fuse-block and *concat* the output of the fuse block with the node features according to the channel. Following the processing of concatenated features by the self-attention mechanism, they are separated according to the corresponding channels to obtain node features and connection features respectively. This process can be described as:

$$f_{fuse}^k = \text{Fuse-Block}(e_1^k, e_2^k) \tag{14}$$

$$f_{self}^k = \text{Self-Attention}(\text{concat}[e_1^k, f_{fuse}^k]) \tag{15}$$

$$e_1^{k+1}, e_2^{k+1} = \text{separate}(f_{self}^k) \tag{16}$$

where k denotes the current number of layers, Fuse-Block denotes one of the two fusion methods introduced in Section 3.3, **concat** denotes splicing node features and connection features according to dimensions. **separate** denotes separating node features and connection features based on dimensions.

3.7 Loss Function

Loss Function of NAS-bench-101 and NAS-bench-201 Since the neural architecture performance predictor can be viewed as a regression model, the straightforward idea is to use the mean squared error (MSE) between the predicted outcome and the actual outcome as the loss function.

$$\text{MSE\_loss}(\hat{y}, y) = \sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{17}$$

However, MSE loss forces models to learn to predict absolute performance rather than relative rankings, which seems too strict to obtain accurate predictions. Based on this observation, we performed processing with MSE loss. We normalize the score of the architecture of the dataset, which is beneficial to increase the gap between the performance value of two networks and increase their related ranking information.

Loss Function of NAS-bench-301 and DARTS Due to the more complex search space of DARTS, it is not possible to obtain the normalization verification accuracy and testing accuracy of all architectures in advance as NAS-bench-101 and NAS-bench-201. We propose to use a ranking-based loss function to replace the direct MSE loss. The rank loss can be described as:

$$\text{rank\_loss}(\hat{y}, y) = \sum_{i=1}^{n}[(\hat{y}_{I(i)} - \hat{y}_i) - (y_{I(i)} - y_i)] \tag{18}$$

where $I(.)$ is the result of random shuffling of sequences 1 to n.

3.8 Search Method

As shown in Algorithm 1, we use our predictor to enhance the predictor-independent search strategy and

Table 1: Comparison with other methods on NAS-Bench-101. We calculate Kendall's Tau by predicting accuracy of all architectures in NAS-Bench-101. "SE" refers to self-evolution proposed by TNASP.

Training samples	100(0.02%)	172(0.04%)	424(0.1%)	424(0.1%)	4236(1%)
Validation samples	200	200	200	200	200
Test samples	*all*	*all*	100	*all*	*all*
Neural predictor (Wen et al., 2020)	0.391	0.545	0.710	0.679	0.769
NAO (Luo et al., 2018)	0.501	0.566	0.704	0.666	0.775
ReNAS (Xu et al., 2021)	-	-	0.634	0.657	0.816
GraphTrans (Wu et al., 2021)	0.330	0.472	0.600	0.602	0.700
Graphormer (Ying et al., 2021)	0.564	0.580	0.596	0.611	0.797
GATES (Ning et al., 2020)	0.605	0.659	0.666	0.691	0.822
GMAE-NAS (Jing et al., 2022)	0.666	0.697	0.788	0.732	0.775
TNASP (Lu et al., 2021)	0.600	0.669	0.752	0.705	0.820
TNASP + SE (Lu et al., 2021)	0.613	0.671	0.754	0.722	0.820
PINAT (Lu et al., 2023)	0.679	0.715	**0.801**	**0.772**	**0.846**
GNET(Node)	0.628	0.730	0.766	0.749	0.834
GNET(Connection)	**0.705**	**0.732**	0.770	0.759	0.842

Table 2: Comparison with other methods on NAS-Bench-201. We calculate Kendall's Tau by predicting accuracy of all architectures in NAS-Bench-201. "SE" refers to self-evolution proposed by TNASP.

Training samples	78(0.5%)	156(1%)	469(3%)	781(5%)	1563(10%)
Validation samples	200	200	200	200	200
Test samples	*all*	*all*	*all*	*all*	*all*
Neural predictor (Wen et al., 2020)	0.343	0.413	0.584	0.634	0.646
NAO (Luo et al., 2018)	0.467	0.493	0.470	0.522	0.526
GraphTrans (Wu et al., 2021)	0.409	0.550	0.594	0.588	0.673
Graphormer (Ying et al., 2021)	0.505	0.630	0.680	0.719	0.776
TNASP (Lu et al., 2021)	0.539	0.589	0.640	0.689	0.724
TNASP + SE (Lu et al., 2021)	0.565	0.594	0.642	0.690	0.726
PINAT (Lu et al., 2023)	0.549	0.631	0.706	0.761	0.784
GNET(Node)	0.559	0.603	0.693	0.718	0.753
GNET(Connection)	**0.578**	**0.634**	**0.723**	**0.762**	**0.794**

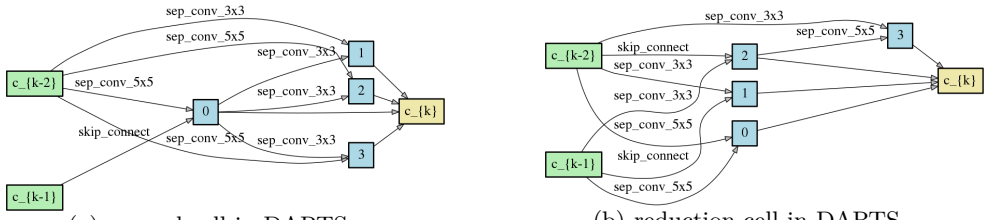
use it for architecture search on DARTS search space. The specific process can be described as follows: use a pre-trained predictor and randomly update the population using an evolutionary algorithm in each iteration to obtain new architectures. After evaluation with the predictor, we select Top-K architectures each time according to the score and add them into population and the architecture pool. Finally, we train the Top-10 architectures in the architecture pool on CIFAR-10 dataset to obtain the optimal neural architecture.

4 EXPERIMENTS

We verify the performance of our method on three different search spaces, including NAS-Bench-101, NAS-bench-201(We use the average of five experimental results for different random number seeds as the final experimental result), and DARTS. The fusion methods of the models we used in the experiments on NAS-bench-101, NAS-bench-201 and DARTS are Cross-Attention and the GNN part is composed of GAT. We compare the performance of two different feature selection methods in NAS-bench-101 and NAS-bench-201, but we only compared the performance of two different feature fusion methods in Ablation Studies. This is because the experimental results of the two different fusion methods are not significantly different. More settings for model training and search hyperparameters will be explained in the supplementary material.

4.1 Experiments on NAS-Bench-101

The NAS-Bench-101 search space consists of 423K unique convolutional architecture and the space is "operation on nodes" (OON). Each architecture is restricted to the cell space. Each cell can have up to 7 nodes and 9 edges. For each node, the candidate



(a) normal cell in DARTS (b) reduction cell in DARTS

Figure 3: Our best searched normal cell and reduction cell.

Algorithm 1: GNET Predictor Based NAS

Input: Search space A, well trained predictor P, the number of iterations T, the number of architectures selected by predictor in each iteration K, the architecture pool S

Initialize the population with random architectures
for $t = 1$ *to* T **do**
  Generate a set of N candidates by randomly mutating the M high-accuracy architectures selected by Tournament in population
  For each candidate $a \in N$, evaluate its performance $P(a)$ by predictor P
  Choose Top-K candidates by $P(a)$ and add them into **population** and architecture pool
  Remove the oldest individual from left of population
end
Train and evaluate the top-10 architectures in S to acquire their actual performance metrics on classification tasks.

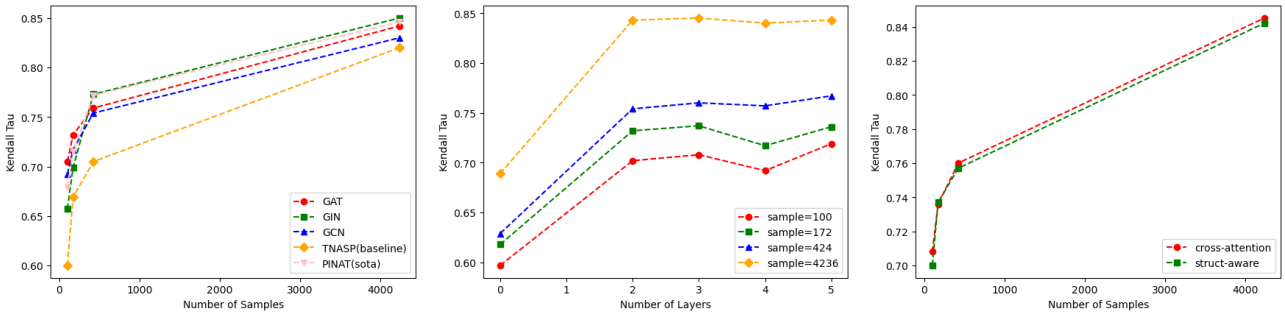
Output: the architecture a with the best true performance.

operations include 1x1 convolution, 3x3 convolution, and 3x3 max pooling. NAS-Bench-101 provides the validation accuracy and the test accuracy on CIFAR-10. Following TNASP, we utilize the validation accuracy from a single run as the training target and apply the mean test accuracy over three runs as the ground truth accuracy for evaluating the performance of our predictions. Refer to settings in related works, we choose 0.02%, 0.04%, 0.1% and 1% of the whole data in search space to train the model as our training data. And we choose 100 samples or all data as the test set. Following existing works, we use Kendall's Tau to evaluate the performance of predictors. In this section, we highlight the distinct roles of the training and test sets. The training set is utilized to refine the predictor's ability to accurately forecast architectural scores. Conversely, the test set serves to evaluate the alignment between the predictor's score prediction distribution across the entire search space and the genuine performance distribution, with Kendall's coefficient as the metric of assessment. This delineation ensures a rigorous examination of the predictor's efficacy. All our experiments are performed on a Tesla V100.

As shown in Table 1, our predictor has significant performance improvements compared to Neural Predictor, NAO, GraphTrans, and Grapher in terms of small data volumes. Some unsupervised methods, such as GMAE, use pre-training method to improve predictor performance when the training set is small, but our method still exceeds them. Although TNASP uses SE method for intensive training, our model still exceeds it, which proves that our model has a stronger ability to obtain architectural representations by using GNN to generate extra information. With the increase of training data (0.1%, 1%), the performance of all models has been significantly improved, but our predictor still maintains the advantage, although the performance is slightly lower than PINAT , thanks to Transformer's powerful long-range modeling capabilities. In Ablation Studies, we use GIN to replace GAT, which compensates for our slightly lower performance than PINAT when training data increases.

4.2 Experiments on NAS-Bench-201

The NAS-Bench-201 search space consists of 15625 unique convolutional architectures and the space is "operation on edges" (OOE). Each architecture is still restricted to the cell space. Each cell is composed of 4 nodes and 6 edges. The candidate operations on each edge include zeroize, skip connection, 1x1 convolution, 3x3 convolution, and 3x3 average pooling. NAS-Bench-201 (Dong and Yang, 2020) provides three different results of each architecture (CIFAR-10, CIFAR-100 and ImageNet). And we choose the validation accuracy and the test accuracy on CIFAR-10. Similar to the experimental setup on NAS-Bench-101, we choose 0.5%,1%,3%,5%, and 10% of the whole data in the search space as our training data to train the model. And we choose all the data as the test set.



(a) Graph network enhancing strategies (b) Numbers of layers (c) Fusion method

Figure 4: Ablation studies

Table 3: Comparison with other methods on DARTS.

Architecture	Top1-Acc(%)
VGG-19 (Simonyan et al., 2015)	95.13
DenseNet-BC (Huang et al., 2017)	96.54
Swin-S (Liu et al., 2021)	94.17
Nest-S (Zhang et al., 2022)	96.97
AmoebaNet-A (Real et al., 2019)	96.66
ENAS (Pham et al., 2018)	97.11
PNAS (Liu et al., 2018)	96.59
GHN (Zhang et al., 2019a)	97.16
D-VAE (Zhang et al., 2019b)	94.80
NGE (Li et al., 2020a)	97.40
BONAS-A (Shi et al., 2020)	97.31
CTNAS (Chen et al., 2021)	97.41
CATE (Yan et al., 2021)	97.45
TNASP (Lu et al., 2021)	97.48
PINAT (Lu et al., 2023)	97.58
GNET	**97.61**

Table 4: Comparison with other methods on NAS-bench-301.

Architecture	Kendall's Tau	Acc
TNASP (Lu et al., 2021)	51.12	0.9402
GNET	**64.41**	**0.9463**

As shown in Table 2, our method outperforms other methods. When the amount of data is small, our method exceeds the current SOTA method by 0.03%, even higher than the TNASP assisted by SE. As TNASP said, when the number of validation samples is greater than the training samples, the predictor can get a larger performance improvement using the SE-framework. This provides evidence that our model can learn a very effective representation of the architecture used for accuracy prediction.

4.3 Experiments on DARTS

The architectures in the DARTS search space are composed of normal cells and reduced cells. Each cell consists of 7 nodes and 14 edges. The DARTS search space is a large space containing approximately 10^{18} structures and is also "operation on edges" (OOE). The candidate operations on each edge include 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. In order to save the training cost of supernet (Lu et al., 2021, 2023), we use the performance of the network architecture on NAS-bench-301 to approximate its true performance on CIFAR-10 dataset. Afterward, we sampled the architecture from NAS-bench-301 to train our predictor and apply it in an evolutionary algorithm to search good architectures in the search space.

As shown in Table 3, we can notice that the best-searched cells get the highest test accuracy 97.61%, which implies that our predictor learns the effective correspondence between architecture and accuracy. We visualize the best searched-architecture in Figure 3. It can be observed that our reduction cell does not include pooling operations, which is rare in the architecture sets searched by other methods. This proves that our predictor can help search strategies find better neural architectures. We also compared our method with TNASP's performance on NAS-bench-301 using 500 samples as the training set. As shown in Table 4, our method has significant advantages over TNASP, which proves that it can adapt to more complex search spaces to achieve better prediction results.

4.4 Ablation Studies

Different Graph Network Enhance Strategies We compare different graph network enhancing strategies on NAS-Bench-101. As shown in Figure 4(a), GAT performs better when data is less, while GIN performs better when data is more. Three different

graph network enhancement methods all significantly outperform TNASP. This validates the effectiveness of our approach in enriching neural architecture representation by integrating extra features via GNN, highlighting its superior performance. Compared with the SOTA method, our method has an advantage when using GAT with less training data (0.02%, 0.04%), and has an advantage when using GIN with more training data (0.1%, 1%).

Different Numbers of Layers The contribution of our method lies in its ability to enhance the representation of neural architectures by fusing GNN features and Transformer features in the fuse block. Here, we attempt to demonstrate that GNN provides crucial predictive information and investigate how the selection of number of layers k affects the results. As shown in Figure 4(b), when $k = 0$, our model is equivalent to a simple transformer. Through the introduction of GNN, the performance of the predictor has significantly improved, which indicates that GNN, when combined with Transformer, does enhance the representation of the neural architecture resulting in a more robust predictor. We choose 3 as the number of layers in our predictor because as the number of layers further increases, the performance of predictor does not change much and the calculation time will increase.

Cross-Attention VS Structure-Aware We compare the two feature fusion methods on NAS-bench-101. As Figure 4(c) shows, the performance difference between the two different feature fusion methods is not significant, but Cross-Attention performs slightly better. Compared to TNASP, the experimental results of these two fusion methods on various benchmarks significantly leads in performance, which demonstrate the effectiveness of both feature fusion approaches in enhancing neural structure representation.

5 CONCLUSION

In this paper, we propose GNET, a Transformer-based neural architecture performance predictor, which combines GNN and Transformer to enhance the representation of neural architectures. Specifically, We propose two feature fusion and two feature selection methods, all of which outperform the baseline methods. Among them, the combination of Cross-Attention and Connection-based exceeds the state-of-the-art method on several benchmarks. We are sure that our predictor can be combined with various search strategies, which will improve the search efficiency and the performance ceiling of the searched architecture. We believe that combining GNNs and Transformers to implement predictors is a promising approach. In future work, we will further explore the design of more effective fusion modules to improve the stability and efficiency of the neural architecture performance predictor.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *Proceedings of International Conference on Learning Representations, ICLR*, 2021.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of AAAI Conference on Artificial Intelligence, AAAI*, pages 3438–3445, 2020.

Dexiong Chen, Leslie O'Bray, and Karsten M. Borgwardt. Structure-aware transformer for graph representation learning. In *Proceedings of International Conference on Machine Learning, ICML*, pages 3469–3489, 2022.

Yaofo Chen, Yong Guo, Qi Chen, Minli Li, Wei Zeng, Yaowei Wang, and Mingkui Tan. Contrastive neural architecture search with neural architecture comparators. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 9502–9511, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pages 4171–4186, 2019.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of International Conference on Learning Representations, ICLR*, 2020.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *CoRR*, 2020.

Jiemin Fang, Yuzhu Sun, Kangjian Peng, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Fast neural network adaptation via parameter remapping and architecture search. In *Proceedings of International Conference on Learning Representations, ICLR*, 2020.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, pages 1024–1034, 2017.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2261–2269, 2017.

Kun Jing, Jungang Xu, and Pengfei Li. Graph masked autoencoder enhanced predictor for neural architecture search. In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI*, pages 3114–3120, 2022.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of International Conference on Learning Representations, ICLR*, 2017.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of AAAI Conference on Artificial Intelligence, AAAI*, pages 3538–3545, 2018.

Wei Li, Shaogang Gong, and Xiatian Zhu. Neural graph embedding for neural architecture search. In *Proceedings of AAAI Conference on Artificial Intelligence, AAAI*, pages 4707–4714, 2020a.

Zhihang Li, Teng Xi, Jiankang Deng, Gang Zhang, Shengzhao Wen, and Ran He. GP-NAS: gaussian process based neural architecture search. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 11930–11939, 2020b.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of European Conference on Computer Vision, ECCV*, pages 19–35, 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *Proceedings of International Conference on Learning Representations, ICLR*, 2019.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of IEEE International Conference on Computer Vision, ICCV*, pages 9992–10002, 2021.

Shun Lu, Jixiang Li, Jianchao Tan, Sen Yang, and Ji Liu. TNASP: A transformer-based NAS predictor with a self-evolution framework. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, pages 15125–15137, 2021.

Shun Lu, Yu Hu, Peihao Wang, Yan Han, Jianchao Tan, Jixiang Li, Sen Yang, and Ji Liu. Pinat: A permutation invariance augmented transformer for nas predictor. In *Proceedings of AAAI Conference on Artificial Intelligence, AAAI*, pages 8957–8965, 2023.

Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. Neural architecture optimization. *CoRR*, 2018.

Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, 2020.

Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *CoRR*, 2021.

Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based NAS. In *Proceedings of European Conference on Computer Vision, ECCV*, pages 189–204, 2020.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *Proceedings of International Conference on Learning Representations, ICLR*, 2020.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of International Conference on Machine Learning, ICML*, pages 4092–4101, 2018.

Ladislav Rampásek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *CoRR*, 2022.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of AAAI Conference on Artificial Intelligence, AAAI*, pages 4780–4789, 2019.

Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, 2020.

Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T. Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with BONAS. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, 2020.

Karen Simonyan, Andrew Zisserman, and no name. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations, ICLR*, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, pages 5998–6008, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proceedings of International Conference on Learning Representations, ICLR*, 2018.

Wei Wen, Hanxiao Liu, Yiran Chen, Hai Helen Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *Proceedings of European Conference on Computer Vision, ECCV*, pages 660–676, 2020.

Colin White, Willie Neiswanger, and Yash Savani. BANANAS: bayesian optimization with neural architectures for neural architecture search. In *Proceedings of AAAI Conference on Artificial Intelligence, AAAI*, pages 10293–10301, 2021.

Zhanghao Wu, Paras Jain, Matthew A. Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, pages 13266–13279, 2021.

Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. Renas: Relativistic evaluation of neural architecture search. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4411–4420, 2021.

Shen Yan, Kaiqiang Song, Fei Liu, and Mi Zhang. CATE: computation-aware neural architecture encoding with transformers. In *Proceedings of International Conference on Machine Learning, ICML*, pages 11670–11681, 2021.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, pages 28877–28888, 2021.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *Proceedings of International Conference on Machine Learning, ICML*, pages 7105–7114, 2019.

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *Proceedings of International Conference on Learning Representations, ICLR*, 2019a.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-VAE: A variational autoencoder for directed acyclic graphs. In *Proceedings of Conference on Neural Information Processing Systems, NeurIPS*, pages 1586–1598, 2019b.

Zizhao Zhang, Han Zhang, Long Zhao, Ting Chen, Sercan Ö. Arik, and Tomas Pfister. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. In *Proceedings of AAAI Conference on Artificial Intelligence, AAAI*, pages 3417–3425, 2022.

Shenghe Zheng, Hongzhi Wang, and Tianyu Mu. Dclp: Neural architecture predictor with curriculum contrastive learning. *CoRR*, 2023.

Instructions for Paper Submissions to AISTATS 2024: Supplementary Materials

1 CODE AND DATA

The code of GNET and the datasets are provided as supplementary materials(CODE FOR MODEL). Due to the page size limitation and the fact that the datasets used in this paper can be easily obtained from GitHub, only the links to the open-source projects corresponding to the datasets are provided(URL FOR DATASET). As mentioned in the main text, the results of all comparative experiments come from their original papers, and the codes and parameters involved are all from corresponding open source projects.

2 COMPLETE METHODS

Due to the page size limitation, we only provide the best model diagram in the main text. As shown in Figure 1, The difference between the two feature fusion methods is that the sources of Q, K, and V are different. The Cross-Attention-based method uses the features generated by GNN as Q, and the output of Self-Attention as K and V. The Structure-Aware-based method uses the features of GNN as Q and K at the same time, and uses the output of Self-Attention only as V. The final performances of the two different methods on NAS-bench-101 are relatively close, proving that both of our proposed feature fusion methods indeed enhance the graph representation of neural architectures.

The difference between the two different feature selection methods is that the input to the fuse block is different. The Node-based method uses the feature e_1 of Node embedding as input, while the Connection-based method takes both the feature e_1 of Node embedding and the feature e_2 of Connection embedding as input, and finally splices e_1 and the output of the block. The final performances of the two different feature selection methods are quite different. We analyze that there are two reasons: one is because that the dimension of the feature is increased after splicing, which increases the parameter amount of the Transformer block and the other reason is that the single sequence after splicing contains both node features and connection features. The ability to handle sequences of variable length and to model long-range dependency will be fully utilized.

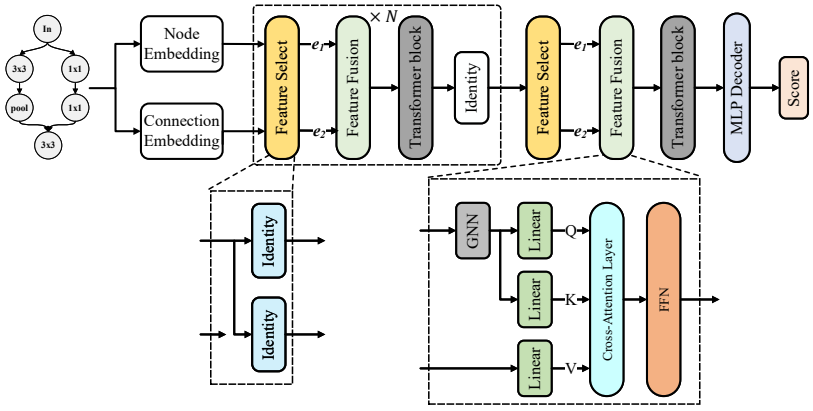
We introduce the configuration information in our training progress. During training, we set the batch size to 10, the epoch to 300, the learning rate to 1e-4, and the dimension of encoder output depends on different methods. Meanwhile, we use an Adam optimizer to optimize the whole network according to the accuracy-rank task. During testing, we set the batch size to 10240 on both NAS-bench-101 and NAS-bench-201.

In addition, the hyperparameter settings of the predictor in search are given in Table 1.

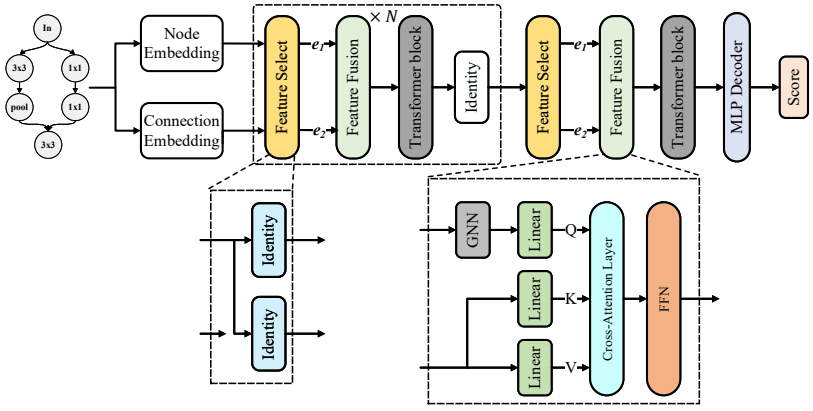
Table 1: The Hyperparameter Settings of Our Model

Parameter	Connection-based	Node-based
GNN layers	3	3
layers	3	3
embedding	80	80
fusion block	80	80
self attention	160	80
MLP	160	80

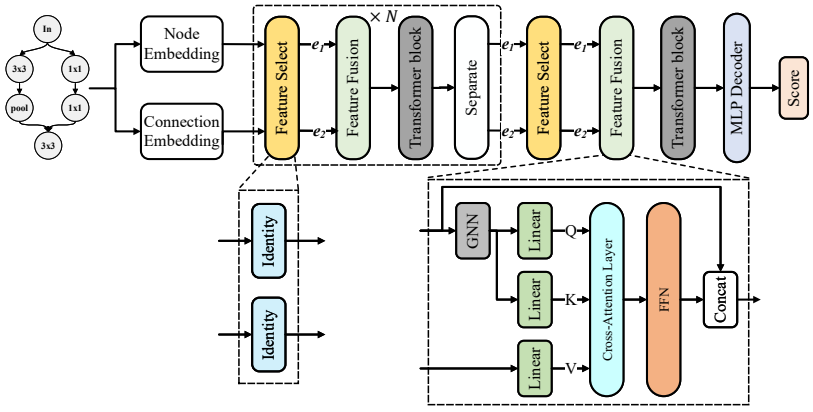
As shown in Table 1, the number of channels in the Self-Attention layer has doubled and the model parameters of the rest of the layers are the same.



(a) Structure-Aware + Node-based



(b) Cross-Attention + Node-based



(c) Structure-Aware + Connection-based

Figure 1: Complete model

3 RETRAINING SETTINGS ON DARTS

DARTS space is more complex than the search space of NAS-bench-101 and NAS-bench-201. For this reason, we cannot use Normalization MSE to train predictors, so we consider using rank loss instead. In the search process, in order to save time and resource consumption, we use the performance of the network architecture on NAS-bench-301 to approximate its true performance on the CIFAR-10 dataset rather than training a supernet. Finally, we retrained the best architecture found in the DARTS search space on CIFAR-10 dataset, and the hyperparameter settings during training follows the DARTS hyperparameter settings shown in Table 2. The training cost is about 2 GPU-days on a Tesla V100.

Table 2: The Hyperparameter Settings on DARTS

Parameter	Connection-based
batch size	96
layers	20
learning rate	0.025
epochs	600
init channels	36
auxiliary weight	0.4
drop path prob	0.2
cutout length	16
grad clip	5

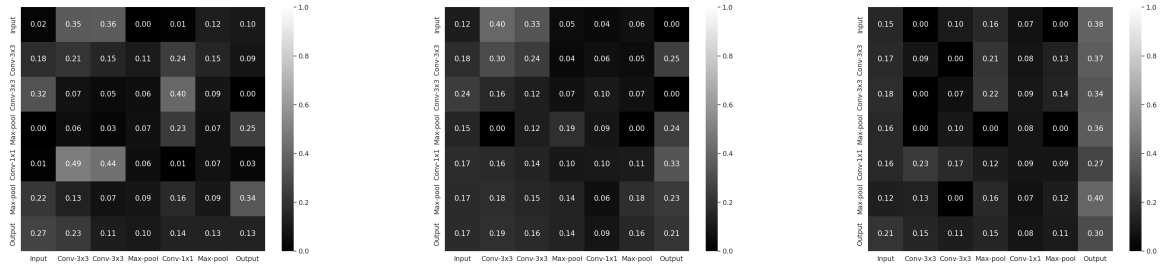
4 ADDITIONAL EXPERIMENTS

4.1 Visualization of Attention Matrix

We compared the attention matrix of TNASP with the attention matrix of our predictor using visualization tools. The colors from light to dark represent the weight of the attention matrix from large to small, ranging from 1 to 0. Both predictors are trained using 100 neural architectures on NAS-bench-101. As shown in Figure 2, the distribution of attention matrix of TNASP is very uneven, this is because that the simple use of the Laplacian matrix as location information lacks the ability to make full use of the structural information of neural architecture. Our method uses GNN to gradually introduce the operation and connection information of the neural architecture, which not only enhances the sequential modeling ability of the Transformer to the neural architecture, but also overcomes the inadequacy of GNN's inability to perform long-distance feature interaction. It can be seen that there are also extreme points in the attention matrix of the first layer, but in the following layers, the features gradually concentrate in the output nodes through the information aggregation effect of GNN. This is more in line with the actual data processing of neural architectures, where the input is passed layer by layer through the neural network, and the information is finally aggregated to the output.

5 DETAILED EXPLANATIONS OF OVER-SMOOTHING PROBLEM

The reason for over-smoothing is that each convolution operation involves the aggregation of information on neighbor nodes, and multiple aggregations may lead to the mixing and loss of information. The representation of nodes will tend to be aggregated, eventually leading to the loss of nodes. Characteristics cannot retain their original distinctiveness. In the neural architecture, the functions of different alternative operation nodes are significantly different, so the distinctiveness of the characteristics of different operation nodes must be ensured.



(a) First layer of our method (b) Second layer of our method (c) Third layer of our method

(d) First layer of TNASP (e) Second layer of TNASP (f) Third layer of TNASP

Figure 2: Visualization of Attention Matrix