FlowerFormer: Empowering Neural Architecture Encoding using a Flow-aware Graph Transformer

Supplementary Material

1. Dataset description

In this section, we provide detailed descriptions of the datasets used.

- **NAS-Bench-101 [17]** is a dataset with 423K architectures trained on the CIFAR-10 [6] dataset. NAS-Bench-101 has an operation-on-node (OON) search space [10, 11]. Following Ning et al. [11], we used the same subset of the NAS-Bench-101 dataset. This subset consists of 14,580 architectures.
- **NAS-Bench-201 [3]** is a dataset with 15K architectures trained on the CIFAR-10 dataset. We transformed the dataset, which is originally operation-on-edge (OOE)-based, into the OON format.
- **NAS-Bench-301 [18]** is a surrogate benchmark with 57K architectures each of which consists of two cells (spec., normal and reduction cells). This dataset is originally OOE-based, and we converted the dataset into the OON format. Following Ning et al. [11], we only used the anchor architecture-performance pairs.
- **NAS-Bench-ASR [9]** is a dataset with 8K architectures of auto speech recognition models, trained on the TIMIT audio dataset [5]. We transformed the dataset, which is originally OOE-based, into the OON format.
- **NAS-Bench-Graph [12]** is a dataset with 26K architectures of graph neural networks, trained on the Cora dataset [14]. Since the dataset is originally OON-based, no additional transformation is required.

2. Experimental Details

2.1. Implementation details

In this subsection, we provide several implementation details of FLOWERFORMER.

Code implementation: We employed the framework of GraphGPS [13] as the backbone to implement FLOWERFORMER with Python 3.10, Pytorch 1.13.1, and Pytorch Geometric 2.2.0.

Obtaining representations in two-cell datasets: To obtain representations in two-cell-based datasets (e.g., NAS-Bench-301), we used the following projection strategy:

Let $h_{1,o} \in \mathbb{R}^d$ and $h_{2,o} \in \mathbb{R}^d$ denote the embeddings of the output nodes of cell 1 and cell 2 after forward message passing. Then, we concatenated $h_{1,o}$ and $h_{2,o}$ and projected the concatenated embeddings with a learnable projection matrix $W^P \in \mathbb{R}^{2d \times 2d}$, as follows:

$$h' = \text{concat}\,(h_{1,o}, h_{2,o})W^{\text{P}}. \tag{1}$$

Then, we split $h' \in \mathbb{R}^{2d}$ into two and regarded each split as an embedding of the output nodes:

$$h_{1,o} = (h'_1, h'_2, \cdots, h'_d), \tag{2}$$
$$h_{2,o} = (h'_{d+1}, h'_{d+2}, \cdots, h'_{2d}), \tag{3}$$

where h'_i is the i-th entry of h'. Finally, we started asynchronous backward message passing (step 3 in Algorithm 1 of the main paper) with updated $h_{1,o}$ and $h_{2,o}$.

Training and hyperparameters: We used the AdamW optimizer [7] to train the models, and the best parameters were selected using early stopping. The hyperparameter search space was as follows:

- lr $\in [10^{-4}, 10^{-2}]$
- weight decay $\in [10^{-10}, 10^{-3}]$
- margin $\in \{0.01, 0.05, 0.1, 0.5, 1.0\}$
- $L \in \{4, 5, 6, 7, 8, 9, 10\}$
- $d \in \{64, 128, 256, 512\}$
- $s \in \{4, 8\}$
- dropout $\in \{0.1, 0.2, 0.3, 0.4, 0.5\}$

Further details regarding hyperparameters, including the best hyperparameter combination in each dataset, are available at http://github.com/y0ngjaenius/CVPR2024_FLOWERFormer.

2.2. Batch operation

Asynchronous message passing inevitably introduces some delay, since each operation should be performed in a sequential manner. In order to accelerate the computation, we employed group-based batch processing. Specifically, we used the topological batching strategy [2, 15], which is specialized in handling asynchronous operations. First, we grouped nodes that belong to the same topological generation and regarded each group as a single batch. Then, instead of updating a representation of a single node at a time, we updated representations of nodes that belong to the same batch simultaneously. Note that this simultaneous update process ensures the same result as updating each node in a batch one by one since the updating processes of nodes in the same topological generation are independent of each other. In this manner, for a one-way message passing, we performed $|\mathcal{T}^G|$ operations, which is generally smaller than the number of nodes.

2.3. Baseline methods

- **GatedGCN [1] and GraphGPS [13]:** We used the GatedGCN implementation provided by the GraphGPS repository. For GraphGPS, we used GatedGCN and Performer as the MPNN and attention modules, respectively. We followed the choice used for OGBG-CODE2, which is the only dataset modeled as a DAG in [13]. The GitHub repository is https://github.com/rampasek/GraphGPS

- **DAGNN [15]:** This model has a bidirectional option, and we considered whether to use it or not as a hyperparameter. The GitHub repository is https://github.com/vthost/DAGNN
- **DAGFormer [8]:** DAGFormer introduces a framework that is applicable to existing graph transformers, We used the DAG+GraphGPS setting, which uses depth positional encoding and replaces the attention module of GraphGPS with reachability attention. The GitHub repository is https://github.com/LUOyk1999/DAGformer
- **NAR-Former [16]:** We followed the augmentation technique and hyperparameter setting of NAR-Former used in [16] for each dataset. The GitHub repository is https://github.com/yuny220/NAR-Former
- **TA-GATES [11]:** We followed the hyperparameter setting of TA-GATES used in [11] for each dataset. While the NAS-Bench-ASR dataset is OOE-based with multi-edges, the original TA-GATES implementation does not support multi-edges. Therefore, we converted the dataset into the OON format. The GitHub repository is https://github.com/walkerning/aw_nas

3. Additional Experiments and Results

3.1. Neural architectures search experiments

To validate the practical utility of FLOWERFORMER, we conduct a series of Neural Architecture Search (NAS) experiments. We employ NPENAS [4] as the backbone search algorithm, using TA-GATES, DAGNN, NAR-Former, and FLOWERFORMER as performance predictors. We follow the experimental setup suggested in [4],with the modification of conducting 100 trials. The results in Figure 1 substantiate FLOWERFORMER's superior performance compared to baseline methods.

3.2. Latency prediction experiments

To measure the encoding quality of FLOWERFORMER in various aspects and validate its effectiveness, we conduct a latency prediction experiment on NAS-Bench-201, comparing FLOWERFORMER with NAR-Former [16]. For this comparison, we utilize Mean Absolute Percentage Error (MAPE) and Error Bound Accuracy (Acc(δ)), the same metrics employed by Yi et al. [16] for latency prediction. As shown in Table 1, FLOWERFORMER outperforms NAR-Former in the latency prediction task.

Figure 1. The average test error of the best neural architectures obtained by the NPENAS algorithm using different performance predictors on the NAS-Bench-101 dataset over 100 trials. The plot shows that FLOWERFORMER consistently outperforms other predictors in achieving lower test error rates, establishing its superiority in guiding the NAS process toward more accurate architectural choices.

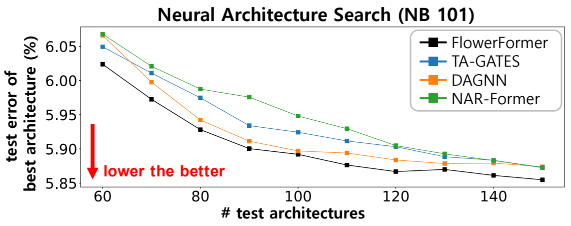

Table 1. Mean Absolute Percentage Error (MAPE) and Error Bound Accuracy (ACC) at δ (scaled up by a factor of 100, mean over 9 trials) of latency prediction on the NAS-Bench-201 dataset. In each setting, the best performances are highlighted in green.

Metric	MAPE↓		ACC ($\delta = 0.1\%$) ↑		ACC ($\delta = 1\%$) ↑		ACC ($\delta = 5\%$) ↑	
Training ratio	5%	10%	5%	10%	5%	10%	5%	10%
NAR-Former	3.1	3.0	2.3	2.3	21.9	22.9	80.8	82.2
FLOWERFORMER	1.1	0.9	8.6	12.7	67.2	78.3	97.4	97.0

3.3. Evaluation with additional metrics

To evaluate the superior performance of FLOWERFORMER across different evaluation criteria, we examine performance on NAS-Bench-101 using the Pearson Coefficient of Linear Correlation (LC) and Root Mean Squared Error (RMSE). As shown in Table 2, FLOWERFORMER shows the best performance in all the settings.

Table 2. Linear Correlation (LC) and Root Mean Squared Error (RMSE) (mean over 9 trials) on the NAS-Bench-101 dataset. In each setting, the best performances are highlighted in green.

Metric	LC↑				RMSE↓			
Training ratio	1%	5%	10%	50%	1%	5%	10%	50%
DAGNN	0.4381	0.4919	0.5201	0.5876	0.0813	0.0802	0.0791	0.0755
TA-GATES	0.3303	0.3432	0.5087	0.5677	0.0834	0.0831	0.0803	0.0777
FLOWERFORMER	0.5636	0.6583	0.6605	0.7483	0.0768	0.0694	0.0670	0.0614

3.4. Extended evaluation on additional dataset

In this section, we analyze the performance of FLOWERFORMER on ENAS, an additional dataset consisting of two cells. As shown in Table 3, FLOWERFORMER achieves the second-best performance in the dataset. We hypothesize that the sub-optimal performance of FLOWERFORMER stems from its failure to account for interactions between two cells. Although there is information flow between two cells, FLOWERFORMER lacks a dedicated global attention module that can capture their interactions. This limitation suggests that enhancing the global attention module to incorporate strategies like cross-attention could be a valuable future research direction.

Table 3. Kendall's Tau (scaled up by a factor of 100, mean over 9 trials) on the ENAS dataset. In each setting, the best performances are highlighted in green.

Datasets	ENAS				Avg.
Training portions	1%	5%	10%	50%	Rank
GatedGCN [1]	15.0	36.1	41.2	54.7	4.75
DAGNN [15]	31.0	47.0	52.6	61.3	1.25
GraphGPS [13]	6.9	26.5	34.2	51.2	6.00
DAGFormer [8]	12.2	41.4	46.5	57.9	4.25
TA-GATES [11]	22.9	45.2	49.4	61.2	2.50
FLOWERFORMER	18.8	44.3	49.5	64.7	2.25

References

[1] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017. 1, 3

[2] Maxwell Crouse, Ibrahim Abdelaziz, Cristina Cornelio, Veronika Thost, Lingfei Wu, Kenneth Forbus, and Achille Fokoue. Improving graph neural network representations of logical formulae with subgraph pooling. *arXiv preprint arXiv:1911.06904*, 2019. 1

[3] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020. 1

[4] C. Wei et al. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 2

[5] John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, and David S Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93:27403, 1993. 1

[6] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 1

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 1

[8] Yuankai Luo, Veronika Thost, and Lei Shi. Transformers over directed acyclic graphs. In *NeurIPS*, 2023. 2, 3

[9] Abhinav Mehrotra, Alberto Gil CP Ramos, Sourav Bhattacharya, Łukasz Dudziak, Ravichander Vipperla, Thomas Chau, Mohamed S Abdelfattah, Samin Ishtiaq, and Nicholas Donald Lane. Nas-bench-asr: Reproducible neural architecture search for speech recognition. In *ICLR*, 2020. 1

[10] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *ECCV*, 2020. 1

[11] Xuefei Ning, Zixuan Zhou, Junbo Zhao, Tianchen Zhao, Yiping Deng, Changcheng Tang, Shuang Liang, Huazhong Yang, and Yu Wang. Ta-gates: An encoding scheme for neural network architectures. In *NeurIPS*, 2022. 1, 2, 3

[12] Yijian Qin, Ziwei Zhang, Xin Wang, Zeyang Zhang, and Wenwu Zhu. Nas-bench-graph: Benchmarking graph neural architecture search. In *NeurIPS*, 2022. 1

[13] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, 2022. 1, 2, 3

[14] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 1

[15] Veronika Thost and Jie Chen. Directed acyclic graph neural networks. In *ICLR*, 2021. 1, 2, 3

[16] Yun Yi, Haokui Zhang, Wenze Hu, Nannan Wang, and Xiaoyu Wang. Nar-former: Neural architecture representation learning towards holistic attributes prediction. In *CVPR*, 2023. 2

[17] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, 2019. 1

[18] Arber Zela, Julien Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks. In *ICLR*, 2022. 1