Appendix for NAS-Bench-Graph: Benchmarking Graph Neural Architecture Search

Yijian Qin, Ziwei Zhang, Xin Wang*, Zeyang Zhang, Wenwu Zhu*
Department of Computer Science and Technology
Tsinghua University
Beijing, China 100084
qinyj19@mails.tsinghua.edu.cn, {zwzhang,xin_wang}@tsinghua.edu.cn
zy-zhang20@mails.tsinghua.edu.cn, wwzhu@tsinghua.edu.cn

A Additional Experimental Details

In this section, we introduce additional experimental details including datasets description, hyper-parameters, and hardware and software configurations.

A.1 Datasets Description

The nine adopted datasets are described in details as follows:

- **Cora**, **Citeseer**, and **Pubmed** [13] are citation graphs where vertices represent papers and links represent citations between papers. Features are bag-of-words and labels are ground-truth topics.
- **Coauthor CS** and **Coauthor Physics** [14] are also co-authorship graphs from Microsoft Academic Graph [15]. Vertices represent authors, links represent co-author relationships, features represent paper keywords of the authors, and vertices labels indicate the research fields of the author.
- **Amazon Computers** and **Amazon Photo** [14] are subsets of co-purchase graph of Amazon [11]. Vertices represent products, links between products represent that they are frequently bought together, features are bag-of-words of product reviews, and vertices labels are the product category.
- **ogbn-arXiv**, a part of the Open Graph Benchmark [8], is a graph representing the citation relationships between papers from arXiv Computer Science (CS) category indexed by MAG [15]. Each paper has a feature vector based on word embedding in its title and abstract. Labels indicate subject areas of papers, and the dataset is split based the chronological order.
- **ogbn-proteins**, also a part of the Open Graph Benchmark [8], is a protein association graph, where vertices represent proteins and links represent different types of biological associations between proteins. The task is to predict the protein functions in a multi-label binary classification setup with data split based on different species the proteins come from.

The datasets are publicly available as follows.

- **Cora**, **Citeseer**, and **Pubmed**: `https://github.com/kimiyoung/planetoid` with MIT Licence.
- **Coauthor CS**, **Coauthor Physics**, **Amazon Computers**, and **Amazon Photo**: `https://github.com/shchur/gnn-benchmark/` with MIT Licence.

*Corresponding Authors.

Submitted to the 36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks. Do not distribute.

- **ogbn-arXiv** and **ogbn-proteins**: `https://ogb.stanford.edu/docs/nodeprop/` with ODC-BY Licence for ogbn-arxiv and CC-0 License for ogbn-proteins.

A.2 Hyper-parameters

The used hyper-parameters are shown in Table 3.

Table 3: The hyper-parameters and hardware used for each dataset. #Pre and #Post denotes the number of pre-process and post-process layers, respectively.

Dataset	#Pre	#Post	Dimension	Dropout	Optimizer	LR	WD	# Epoch
Cora	0	1	256	0.7	SGD	0.1	0.0005	400
CiteSeer	0	1	256	0.7	SGD	0.2	0.0005	400
PubMed	0	0	128	0.3	SGD	0.2	0.0005	500
Coauthor-CS	1	0	128	0.6	SGD	0.5	0.0005	400
Coauthor-Physics	1	1	256	0.4	SGD	0.01	0	200
Amazon-Photo	1	0	128	0.7	Adam	0.0002	0.0005	500
Amazon-Computers	1	1	64	0.1	Adam	0.005	0.0005	500
ogbn-arxiv	0	1	128	0.2	Adam	0.002	0	500
ogbn-proteins	1	1	256	0	Adam	0.01	0.0005	500

A.3 Hardware and Software Configurations

- Operating System: Ubuntu 18.04.6 LTS for PubMed, ogbn-arXiv, and CentOS Linux release 7.6.1810 for the others.
- CPU: Intel(R) Xeon(R) Gold 6129 CPU @ 2.30GHz for PubMed, ogbn-arXiv, and Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz for the others.
- GPU: NVIDIA GeForce RTX 3090 with 24GB of memories for PubMed, ogbn-arXiv, and NVIDIA Tesla V100 with 16GB of memories for the others.
- Software: Python 3.9.12, PyTorch 1.11.0+cu113, PyTorch-Geometric 2.0.4 [6].

A.4 Training Time

Table 4: The average training time of architectures on each dataset.

Dataset	Time	Dataset	Time	Dataset	Time
Cora	5.8s	Coauthor-CS	8.6s	Amazon-Computers	9.8s
CiteSeer	6.2s	Coauthor-Physics	15.4s	ogbn-arXiv	71s
PubMed	7.8s	Amazon-Photo	8.8s	ogbn-proteins	50min

We report the average training time of architectures on each dataset in Table 4. The total time cost of creating our benchmark is approximately 8,000 GPU hours.

A.5 Size of the Search Space

In our proposed search space, we have nine candidate operations for four edges in the DAG, so there are 9^4 operation choices for each type of the macro space. Since we have nine types of macro spaces, we have $9^4 * 9 = 59,049$ architectures in total. However, some of these architectures are isomorphic, e.g., an identity operation followed by GCN layer is equivalent to a GCN layer followed by an identity operation. We obtain 26,206 unique architectures after removing these duplicates.

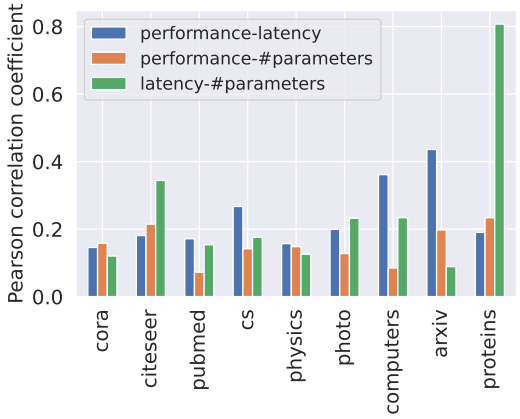
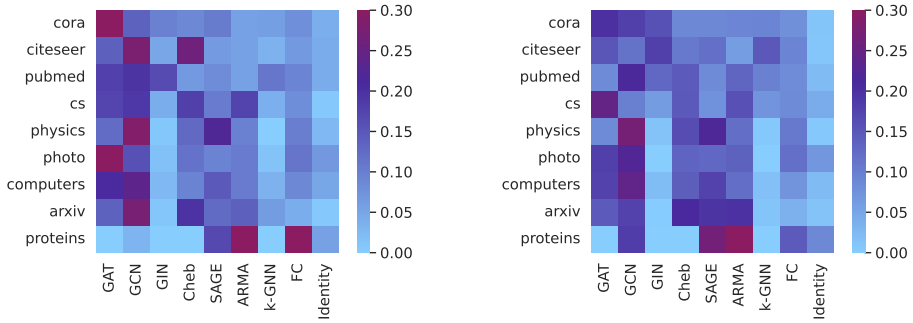


Figure 7: Pearson correlation coefficient between pairs among performance, latency, and the number of parameters.



(a) The frequency of operation choices at depth one.

(b) The frequency of operation choices at depth larger than one.

Figure 8: The frequency of operation choices at different depths in the top 5% architectures of different datasets.

B Additional Experimental Results

B.1 Correlations between Performance, Latency, and the Number of Parameters

We show the Pearson correlation coefficient between the performance, the latency, and the number of parameters in Figure 7. We find that: 1) the number of parameters and the latency are positively correlated in general. 2) Better performance in general also means a larger latency and more parameters, which makes it necessary to balance the performance with the computational overhead.

B.2 The Operation Distribution of Different Depths

To further analyze the operation choices, we plot the frequency of operations with different depths, i.e., the distance of the computation node to the input node. For example, if an operation receives the raw input or the output of the pre-process layer, it is at the depth one; if an operation receives the output of the above operation, it is at depth two, etc. We categorize the operations into two groups, depth one and depth larger than one, and visualize the frequency of operation choices in Figure 8.

The results show that the distribution of operations at different depths shows different patterns on different datasets. For example, for CiteSeer, GCN appears more frequently at depth one, but the opposite for ogbn-proteins. The results indicate that searching for different operations at different depths is critical and simply stacking the same operation cannot lead to the optimal results.

B.3 Transferability of the Optimal Architecture

To investigate the transferability of optimal architectures for different datasets, we calculate the rate at which an optimal architecture obtained on a search dataset outperforms other architectures on another evaluation dataset. The results are shown in Figure 9. We find that in some cases, the optimal architecture can transfer well, e.g., Citeseer to Cora and Physics to arXiv. Besides, the transferability can be asymmetric in some cases, e.g., the optimal architecture on Amazon-Computers performs well on Cora, CiteSeer, PubMed, and Coauthor-CS, but the optimal architectures on these four datasets cannot perform well on Amazon-Computers.

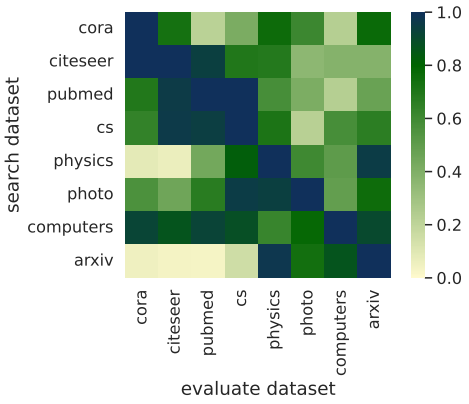


Figure 9: The rate at which the optimal architecture on the search dataset outperforms other architectures on the evaluation dataset.

C Example Usage & Reproducibility

C.1 Example Usage

All our codes and recorded metrics for the trained models are available at `https://github.com/THUMNLab/NAS-Bench-Graph`. Next, we provide some example usages.

At first, the benchmark of a certain dataset, e.g., Cora, can be read as:

```
from readbench import lightread
bench = lightread('cora')
```

The data is stored as a Python dictionary. To obtain the recorded metrics, an architecture needs to be specified by its macro space and operations. Since we constrain the DAG of the computation graph to have only one input node for each intermediate node, the macro space can be described by a list of integers, indicating the input node index for each computing node (0 for the raw input, 1 for the first computing node, etc.). Then, the operations can be specified by a list of strings with the same length. For example, to specify the architecture shown in Figure 10, we can use the following code:

```
from hpo import Arch
arch = Arch([0, 1, 2, 1], ['gcn', 'gin', 'fc', 'cheb'])
```

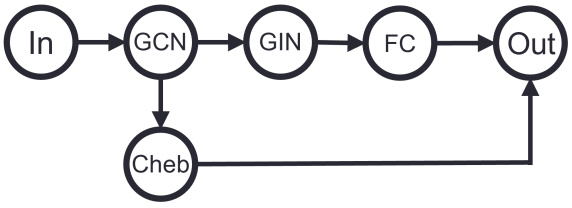


Figure 10: An example architecture.

Notice that we assume all leaf nodes (i.e., nodes without descendants) are connected to the output, so there is no need to specific the output node. Besides, the list can be specified in any order, e.g., the following code can specific the same architecture:

```
arch = Arch([0, 1, 1, 2], ['gcn', 'cheb', 'gin', 'fc'])
```

Then, four recorded metrics in the benchmark including the validation and test performance, the latency, and the number of parameters, can be obtained by a look-up table:

```
info = bench[arch.valid_hash()]
```

```
info['valid_perf']  # validation performance
info['perf']        # test performance
info['latency']     # latency
info['para']        # number of parameters
```

We provide the full data, including the training/validation/testing performance at each epoch at: `https://figshare.com/articles/dataset/NAS-bench-Graph/20070371`. Since we run each dataset with three random seeds, each dataset has 3 files. The full metric can be obtained similarly as follows:

```
from readbench import read
bench = read('cora0.bench')  # dataset and seed
info = bench[arch.valid_hash()]
epoch = 50
info['dur'][epoch][0]   # training performance
info['dur'][epoch][1]   # validation performance
info['dur'][epoch][2]   # testing performance
info['dur'][epoch][3]   # training loss
info['dur'][epoch][4]   # validation loss
info['dur'][epoch][5]   # testing loss
info['dur'][epoch][6]   # best performance
```

We have also provided the source codes of using our benchmark together with two public libraries for GraphNAS, AutoGL and NNI. See `https://github.com/THUMNLab/AutoGL/tree/agnn` and `https://github.com/THUMNLab/NAS-Bench-Graph/blob/main/runnni.py` for details.

C.2 Reproducibility

We have released the source code for our benchmark as well as the detailed hyper-parameters. We have also provided the full metrics of all the architectures in our search space, as discussed in Appendix C.1.

D Discussions

D.1 Preliminary on Graph NAS

Graphs are natural data structures to represent entities and their relationships, with real-world examples including social networks, e-commerce networks, biochemistry, etc. Graph neural networks (GNNs) are de facto standards in processing the ubiquitous graph data. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the vertex set and $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ denotes the link set. The neighborhood of a node v is denoted as $\mathcal{N}(v)$. The vertices are also associated with a feature matrix $\mathbf{F}$. Graph neural networks follow a neighborhood aggregation scheme. At each layer, the representations of nodes are generated through aggregating their neighborhood representations as follows:

$$\begin{aligned} \mathbf{m}_i^{(l)} &= \text{AGGREGATE}\left(\left\{\mathbf{h}_j^{(l)}, \forall h \in \mathcal{N}(i)\right\}\right) \\ \mathbf{h}_i^{(l+1)} &= \text{COMBINE}\left(\left[\mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)}\right]\right), \end{aligned} \tag{1}$$

where $\mathbf{h}_i^{(l)}$ denotes the vertex representation of v_i in the l^{th} layer, $\mathbf{m}_i^{(l)}$ is the message vector, and AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$ are learnable functions. The vertex representations are initialized as vertex features, i.e., $\mathbf{h}_i^{(0)} = \mathbf{F}_{i,:}$. After L message-passing layers, the representation of vertices $\mathbf{H}^{(L)}$ can capture both structural and semantic information within the nodes' L-hop neighborhood. GNNs have achieved the state-of-the-art results for graph analytical tasks such as node classification, link prediction, and graph classification.

Graph neural architecture search aims to further automate the design of GNN architectures, e.g., the operations choices in the learnable functions AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$. The objective of

GraphNAS can be summarized as the following bi-level optimization problem:

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \mathcal{L}_{val}\left(\mathbf{W}^*(\alpha), \alpha\right) \\ \text{s.t.} \quad &\mathbf{W}^*(\alpha) = \arg\min_{\mathbf{W}} \left(\mathcal{L}_{train}\left(\mathbf{W}, \alpha\right)\right), \end{aligned} \tag{2}$$

where α denotes the GNN architecture, $\mathcal{A}$ is the search space, $\mathbf{W}$ are learnable parameters for GNNs, and $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ denotes the training and validation loss, respectively. In short, the goal of GraphNAS is to find the best performing architecture in the search space so that the architecture can achieve the best validation performance after its parameters are trained in the training set.

D.2 A Comparison with the existing NAS benchmarks

Table 5: A comparison with the exsiting NAS benchmarks

Benchmark	Type	Search Space	Data	Datasets
NAS-Bench-101 [17]	Tabular	423k	CV	1
NAS-Bench-201 [4]	Tabular	6k	CV	3
NAS-Bench-1shot1 [19]	Tabular	364k	CV	1
NAS-Bench-ASR [12]	Tabular	8k	Acoustics	1
NAS-Bench-NLP [9]	Tabular	14k	NLP	2
HW-NAS-Bench [10]	Tabular	6k	CV	3
NATS-Bench [3]	Tabular	32k	CV	3
NAs-HPO-Bench-II [7]	Surrogate	192k	CV	1
NAS-Bench-MR [2]	Surrogate	10^{23}	CV	4
TransNAS-Bench [5]	Tabular	7k	CV	14
NAS-Bench-111 [16]	Surrogate	423k	CV	1
NAS-Bench-311 [16]	Surrogate	10^{18}	CV	1
NAS-Bench-Zero [1]	Tabular	34k	CV	3
Surr-NAS-Bench-FBNet [20]	Surrogate	10^{21}	CV	2
NAS-Bench-Graph	Tabular	26k	Graph	9

We provide more comparisons of our proposed benchmark with the existing NAS benchmarks in Table 5. NAS benchmarks can be mainly divided into tabular benchmarks and surrogate benchmarks. In tabular benchmarks, all architectures in the search space are trained to get the empirical performance. On the other hand, surrogate benchmarks use surrogate functions to predict the performance of the architectures. Tabular benchmarks have better authenticity since the results are from experiments, but running experiments can cost lots of computational resources and potentially limits the size of the search space. Surrogate benchmarks are more efficient, but the quality of the benchmark depends highly on the surrogate function. Despite the efforts in creating NAS benchmarks for other domains, e.g., computer vision, NAS benchmark for graphs has not been studied in the literature.

D.3 Comparison with GraphGym

GraphGym [18] is a pioneering work studying the design space of GNNs, which we have drawn inspirations in developing our benchmark. However, GraphGym is a public library and codebase for GNNs, which is different from our proposed benchmark in the following two aspects. First, GraphGym focuses on the search space and does not consider the search strategy of GNN architectures. Second and more importantly, we have trained and provided the performance of all possible architectures in our search space, consuming 8,000 GPU hours. Then, the evaluation of GraphNAS can be obtained by look-up tables for extremely efficient comparisons.

E Broader Impact

GraphNAS can be widely applied to various domains such as social networks, recommendation systems, biological networks, the World Wide Web, etc. Our proposed benchmark can enable fair,

reproducible, and efficient comparisons of GraphNAS methods and therefore promotes further this research direction and benefits the above applications. As for ethical aspects, we do not foresee that our benchmark should produce any biased or offensive content. Besides, our proposed benchmark is based on existing publicly available graph datasets, which do not contain personally identifiable or privacy-related information, to the best of our knowledge.

References

[1] Hanlin Chen, Ming Lin, Xiuyu Sun, and Hao Li. Nas-bench-zero: A large scale dataset for understanding zero-shot neural architecture search. In *OpenReview*, 2021.

[2] Mingyu Ding, Yuqi Huo, Haoyu Lu, Linjie Yang, Zhe Wang, Zhiwu Lu, Jingdong Wang, and Ping Luo. Learning versatile neural architectures by propagating network codes. *arXiv preprint arXiv:2103.13253*, 2021.

[3] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[4] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2019.

[5] Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5251–5260, 2021.

[6] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[7] Yoichi Hirose, Nozomu Yoshinari, and Shinichi Shirakawa. Nas-hpo-bench-ii: A benchmark dataset on joint optimization of convolutional neural network architecture and training hyperparameters. In *Asian Conference on Machine Learning*, pages 1349–1364. PMLR, 2021.

[8] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Neural Information Processing Systems*, 2020.

[9] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *arXiv preprint arXiv:2006.07116*, 2020.

[10] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2020.

[11] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[12] 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 *International Conference on Learning Representations*, 2020.

[13] 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.

[14] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

[15] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020.

[16] Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. *Advances in Neural Information Processing Systems*, 34, 2021.

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

[18] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[19] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2019.

[20] Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate nas benchmarks: Going beyond the limited search spaces of tabular nas benchmarks. In *International Conference on Learning Representations*, 2021.