AutoGL: A Library for Automated Graph Learning

Chaoyu Guan[*], **Ziwei Zhang**[*], **Haoyang Li, Heng Chang, Zeyang Zhang, Yijian Qin, Jiyan Jiang, Xin Wang**[†], **Wenwu Zhu**[†]
Department of Computer Science and Technology, Tsinghua University
`autogl@tsinghua.edu.cn`

Abstract

Recent years have witnessed an upsurge of research interests and applications of machine learning on graphs. Automated machine learning (AutoML) on graphs is on the horizon to automatically design the optimal machine learning algorithm for a given graph task. However, none of the existing libraries can fully support AutoML on graphs. To fill this gap, we present Automated Graph Learning (AutoGL), the first library for automated machine learning on graphs. AutoGL is **open-source**, **easy to use**, and **flexible to be extended**. Specifically, we propose an automated machine learning pipeline for graph data containing four modules: auto feature engineering, model training, hyper-parameter optimization, and auto ensemble. For each module, we provide numerous state-of-the-art methods and flexible base classes and APIs, which allow easy customization. We further provide experimental results to showcase the usage of our AutoGL library.

1 Introduction

Machine learning on graphs has drawn increasing popularity in the last decade (Zhang et al., 2020c). For example, graph neural networks (GNNs) (Zhou et al., 2018; Wu et al., 2020) are the rising star and have shown successes in a wide range of applications such as recommendation (Ma et al., 2019; Li et al., 2021), traffic forecasting (Jiang & Luo, 2021), physical simulations (Shlomi et al., 2020), geometric data analysis (Bronstein et al., 2017), bioinformatics (Su et al., 2020), and combinatorial optimization (Bengio et al., 2020).

However, as the literature booms and graph tasks become ever more diverse, it becomes increasingly difficult to manually design the optimal machine learning algorithm for a given graph task. Therefore, there is an urgent need and recent research interest in automated machine learning (AutoML) on graphs (Zhang et al., 2021). Essentially, AutoML on graphs combines the strengths of graph-based machine learning and AutoML techniques (Yao et al., 2018) to automate the design of graph-based machine learning algorithms. Initial successes have been shown in hyper-parameter optimization (HPO) (Tu et al., 2019) and neural architecture search (NAS) (Gao et al., 2020) for graph learning algorithms.

Public libraries are critical to facilitate and advance the research and applications of AutoML on graphs. Several libraries and toolkits exist for machine learning on graphs, such as PyTorch Geometric (Fey & Lenssen, 2019), Deep Graph Library (Wang et al., 2019), GraphNets (Battaglia et al., 2018), AliGraph (Zhu et al., 2019), and PBG (Lerer et al., 2019). Besides, AutoML libraries such as AutoKeras (Jin et al., 2019), AutoSklearn (Feurer et al., 2019), Hyperopt (Bergstra et al., 2013), and NNI (Zhang et al., 2020a) are also available. Unfortunately, integrating these libraries is non-trivial due to the challenges of AutoML on graphs (see (Zhang et al., 2021) for a comprehensive survey). Currently, there are no public libraries for AutoML on graphs, to the best of our knowledge.

To tackle this problem, we present Automated Graph Learning (AutoGL), the first dedicated framework and library for automated machine learning on graphs. The overall framework of AutoGL is

[*]Equal contributions.
[†]Corresponding authors.

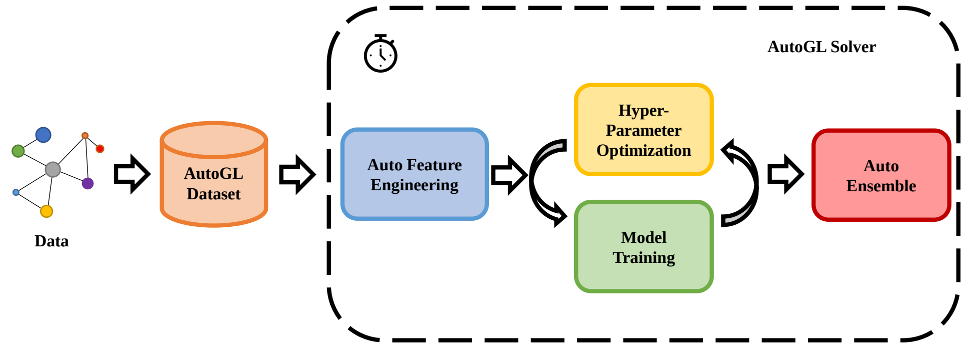

Figure 1: The overall framework of AutoGL.

shown in Figure 1. We summarize and abstract the pipeline of AutoML on graphs into four modules: auto feature engineering, model training, hyper-parameter optimization, and auto ensemble. For each module, we provide plenty of state-of-the-art algorithms, standardized base classes, and high-level APIs for easy and flexible customization. The AutoGL library is built upon PyTorch Geometric (PyG) (Fey & Lenssen, 2019), a widely adopted graph machine learning library. AutoGL has the following key characteristics:

- **Open source:** The code[1] and detailed documentation[2] are available online.
- **Easy to use:** AutoGL is designed to be user-friendly. Users can conduct quick AutoGL experiments with less than ten lines of code.
- **Flexible to be extended:** The modular design, high-level base class APIs, and extensive documentation of AutoGL allow flexible and easy customized extensions.

2 Automated Graph Learning

In this section, we introduce AutoGL designs in detail. AutoGL is designed in a modular and object-oriented fashion to enable clear logic flows, easy usages, and flexible extensions. All the APIs exposed to users are abstracted in a high-level fashion to avoid redundant re-implementation of models, algorithms, and train/evaluation protocols. As illustrated in Figure 1, we summarize and abstract the core problem of AutoML on graphs into a pipeline with four modules: auto feature engineering, model training, hyper-parameter optimization, and auto ensemble. All these modules have taken into account the unique characteristics of machine learning on graphs.

AutoGL Dataset We first briefly introduce our dataset management. AutoGL Dataset is currently based on `Dataset` from PyTorch Geometric and supports common benchmarks for node and graph classification, including the recent Open Graph Benchmark (Hu et al., 2020). For a complete list of datasets, please refer to Appendix A. Users can also easily customize datasets following our documentation.

Auto Feature Engineering The graph data is first processed by the auto feature engineering module, where various nodes, edges, and graph-level features can be automatically added, compressed, or deleted to help boost the graph learning process after. Graph topological features can also be extracted to utilize graph structures better.

Currently, we support 24 feature engineering operations abstracted into three categories: generators, selectors, and graph features. The generators aim to create new node and edge features based on the current node features and graph structures. The selectors automatically filter out and compress features to ensure they are compact and informative. Graph features focus on generating graph-level features. For a complete list, please refer to Appendix B. We also provide convenient wrappers that support feature engineering operations in PyTorch Geometric (Fey &

[1]`https://github.com/THUMNLab/AutoGL/`
[2]`https://autogl.readthedocs.io/`

Table 1: The results of node classification

Model	Cora	CiteSeer	PubMed
GCN	80.9 ± 0.7	70.9 ± 0.7	78.7 ± 0.6
GAT	82.3 ± 0.7	71.9 ± 0.6	77.9 ± 0.4
GraphSAGE	74.5 ± 1.8	67.2 ± 0.9	76.8 ± 0.6
AutoGL	$\mathbf{83.2 \pm 0.6}$	$\mathbf{72.4 \pm 0.6}$	$\mathbf{79.3 \pm 0.4}$

Table 2: The results of graph classification

Model	MUTAG	PROTEINS	IMDB-B
Top-K Pooling	80.8 ± 7.1	69.5 ± 4.4	71.0 ± 5.5
GIN	82.7 ± 6.9	66.5 ± 3.9	69.1 ± 3.7
AutoGL	$\mathbf{87.6 \pm 6.0}$	$\mathbf{73.3 \pm 4.4}$	$\mathbf{72.1 \pm 5.0}$

Table 3: The results of different HPO methods for node classification

		Cora		CiteSeer		PubMed	
Method	Trials	GCN	GAT	GCN	GAT	GCN	GAT
None		80.9 ± 0.7	82.3 ± 0.7	70.9 ± 0.7	71.9 ± 0.6	78.7 ± 0.6	77.9 ± 0.4
random	1	81.0 ± 0.6	81.4 ± 1.1	70.4 ± 0.7	70.1 ± 1.1	78.3 ± 0.8	76.9 ± 0.8
	10	82.0 ± 0.6	82.5 ± 0.7	71.5 ± 0.6	$\mathbf{72.2 \pm 0.7}$	79.1 ± 0.3	78.2 ± 0.3
	50	81.8 ± 1.1	$\mathbf{83.2 \pm 0.7}$	71.1 ± 1.0	72.1 ± 1.0	$\mathbf{79.2 \pm 0.4}$	78.2 ± 0.4
TPE	1	81.8 ± 0.6	81.9 ± 1.0	70.1 ± 1.2	71.0 ± 1.2	78.7 ± 0.6	77.7 ± 0.6
	10	82.0 ± 0.7	82.3 ± 1.2	71.2 ± 0.6	72.1 ± 0.7	79.0 ± 0.4	$\mathbf{78.3 \pm 0.4}$
	50	$\mathbf{82.1 \pm 1.0}$	83.2 ± 0.8	$\mathbf{72.4 \pm 0.6}$	71.6 ± 0.8	79.1 ± 0.6	78.1 ± 0.4

Lenssen, 2019) and NetworkX (Hagberg et al., 2008). Users can easily customize feature engineering methods by inheriting from the class `BaseGenerator`, `BaseSelector`, and `BaseGraph`, or `BaseFeatureEngineer` if the methods do not fit in our categorization.

Model Training This module handles the training and evaluation process of graph machine learning tasks with two functional sub-modules: Model and Trainer. Model handles the construction of graph machine learning models, e.g., GNNs, by defining learnable parameters and the forward pass. Trainer controls the optimization process for the given model. Common optimization methods are packaged as high-level APIs to provide neat and clean interfaces. More advanced training controls and regularization methods in graph tasks like early stopping and weight decay are also supported.

The model training module supports both node-level and graph-level tasks, e.g., node classification and graph classification. Commonly used models for node classification such as GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), and GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019), and pooling methods such as Top-K Pooling (Gao & Ji, 2019) are supported. Users can quickly implement their own graph models by inheriting from the `BaseModel` class and add customized tasks or optimization methods by inheriting from `BaseTrainer`.

Hyper-Parameter Optimization The HPO module aims to automatically search for the best hyper-parameters of a specified model and training process, including but not limited to architecture hyper-parameters such as the number of layers, the dimensionality of node representations, the dropout rate, the activation function, and training hyper-parameters such as the learning rate, the weight decay, the number of epochs. The hyper-parameters, their types (e.g., integer, numerical, or categorical), and feasible ranges can be easily set.

We have supported various HPO algorithms, including algorithms specified for graph data like AutoNE (Tu et al., 2019) and general-purpose algorithms like random search (Bergstra & Bengio, 2012), Tree Parzen Estimator (Bergstra et al., 2011), etc. Users can customize HPO algorithms by inheriting from the `BaseHPOptimizer` class.

Auto Ensemble This module can automatically integrate the optimized individual models to form a more powerful final model. Currently, we have adopted two kinds of ensemble methods: voting and stacking. Voting is a simple yet powerful ensemble method that directly averages the output of individual models. Stacking trains another meta-model to combine the output of models. We have supported general linear models (GLM) and gradient boosting machines (GBM) as meta-models.

AutoGL Solver On top of the modules mentioned above, we provide another high-level API Solver to control the overall pipeline. In Solver, the four modules are integrated systematically to form the final model. Solver receives the feature engineering module, a model list, the HPO module, and the ensemble module as initialization arguments to build an Auto Graph Learning pipeline. Given a dataset and a task, Solver first perform auto feature engineering to clean and augment the

Table 4: The results of different HPO methods for graph classification

HPO	MUTAG		PROTEINS		IMDB-B	
	Top-K Pooling	GIN	Top-K Pooling	GIN	Top-K Pooling	GIN
None	76.3 ± 7.5	82.7 ± 6.9	69.5 ± 4.4	66.5 ± 3.9	71.0 ± 5.5	69.1 ± 3.7
random	82.7 ± 6.8	$\mathbf{87.6 \pm 6.0}$	$\mathbf{73.3 \pm 4.4}$	$\mathbf{71.0 \pm 5.9}$	71.5 ± 4.1	$\mathbf{71.3 \pm 4.0}$
TPE	$\mathbf{83.9 \pm 10.1}$	86.7 ± 6.2	72.3 ± 5.5	71.0 ± 7.2	$\mathbf{71.6 \pm 2.5}$	70.2 ± 3.7

input data, then optimize all the given models using the model training and HPO module. At last, the optimized best models will be combined by the Auto Ensemble module to form the final model.

Solver also provides global controls of the AutoGL pipeline. For example, the time budget can be explicitly set to restrict the maximum time cost, and the training/evaluation protocols can be selected from plain dataset splits or cross-validation.

3 EVALUATION

In this section, we provide experimental results. Note that we mainly want to showcase the usage of AutoGL and its main functional modules rather than aiming to achieve the new state-of-the-art on benchmarks or compare different algorithms. For node classification, we use Cora, CiteSeer, and PubMed with the standard dataset splits from (Kipf & Welling, 2017). For graph classification, we follow the setting in (Errica et al., 2020) and report the average accuracy of 10-fold cross-validation on MUTAG, PROTEINS, and IMDB-B.

Table 5: The performance of the ensemble module of AutoGL for the node classification task.

Base Model	Cora	CiteSeer	PubMed
GCN	81.1 ± 0.9	69.6 ± 1.1	$\mathbf{78.5 \pm 0.4}$
GAT	82.0 ± 0.5	70.4 ± 0.6	77.7 ± 0.5
Ensemble	$\mathbf{82.2 \pm 0.4}$	$\mathbf{70.8 \pm 0.5}$	$\mathbf{78.5 \pm 0.4}$

AutoGL Results We turn on all the functional modules in AutoGL, and report the fully automated results in Table 1 and Table 2. We use the best single model for graph classification under the cross-validation setting. We observe that in all the benchmark datasets, AutoGL achieves better results than vanilla models, demonstrating the importance of AutoML on graphs and the effectiveness of the proposed pipeline in the released library.

Hyper-Parameter Optimization Table 3 reports the results of two implemented HPO methods, i.e., random search and TPE (Bergstra et al., 2011), for the semi-supervised node classification task. As shown in the table, as the number of trials increases, both HPO methods tend to achieve better results. Besides, both methods outperform vanilla models without HPO. Note that a larger number of trials do not guarantee better results because of the potential overfitting problem. We further test these HPO methods with ten trials for the graph classification task and report the results in Table 4. The results generally show improvements over the default hand-picked parameters on all datasets.

Auto Ensemble Table 5 reports the performance of the ensemble module as well as its base learners for the node classification task. We use voting as the example ensemble method and choose GCN and GAT as the base learners. The table shows that the ensemble module achieves better performance than both the base learners, demonstrating the effectiveness of the implemented ensemble module.

4 CONCLUSION AND FUTURE PLANS

We have presented AutoGL, the first library for automated machine learning on graphs, which is open-source, easy to use, and flexible to be extended. Currently, we are actively developing AutoGL and plan to support the following functionalities within a short time:

- Support neural architecture search.
- Support for large-scale graphs.
- Handle more graph tasks, e.g., heterogeneous graphs and spatial-temporal graphs.
- Support more graph library backends, e.g., Deep Graph Library (Wang et al., 2019).

All kinds of inputs and suggestions are also warmly welcomed.

Acknowledgments

The authors thank anonymous reviewers for their detailed comments. The authors also thank Zixin Sun (SIGS, Tsinghua University) for his feedbacks on AutoGL. This work is supported in part by the National Key Research and Development Program of China (No.2020AAA0106300, 2020AAA0107800, 2018AAA0102000) and National Natural Science Foundation of China No.62050110.

References

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24, 2011.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pp. 115–123, 2013.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21:i47–i56, 2005.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations (ICLR 2020)*, 2020.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pp. 113–134. Springer, Cham, 2019.

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

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 1403–1409, 7 2020.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, 2008.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

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. *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *arXiv preprint arXiv:2101.11174*, 2021.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956, 2019.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

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

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, 2019.

Haoyang Li, Xin Wang, Ziwei Zhang, Jianxin Ma, Peng Cui, and Wenwu Zhu. Intention-aware sequential recommendation with structured intent transition. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Jianxin Ma, Chang Zhou, Peng Cui, Hongxia Yang, and Wenwu Zhu. Learning disentangled representations for recommendation. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. Deep inductive graph representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 32(3):438–452, 2018.

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.

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

Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2020.

Chang Su, Jie Tong, Yongjun Zhu, Peng Cui, and Fei Wang. Network embedding in biomedical data science. *Briefings in bioinformatics*, 21(1):182–197, 2020.

Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2347–2356, 2018.

Ke Tu, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu. Autone: Hyperparameter optimization for massive network embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 216–225, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv:1909.01315*, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, and Lidong Zhou. Retiarii: A deep learning exploratory-training framework. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pp. 919–936, 2020a.

Ziwei Zhang, Peng Cui, Jian Pei, Xin Wang, and Wenwu Zhu. Eigen-gnn: A graph structure preserving plug-in for gnns. *arXiv:2006.04330*, 2020b.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020c.

Ziwei Zhang, Xin Wang, and Wenwu Zhu. Automated machine learning on graphs: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2021. Survey track.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: A comprehensive graph neural network platform. *Proc. VLDB Endow.*, 12 (12):2094–2105, 2019.

A Datasets

Currently, we provide widely adopted node classification datasets including Cora, CiteSeer, PubMed (Sen et al., 2008), Amazon Computers, Amazon Photo, Coauthor CS, Coauthor Physics (Shchur et al., 2018), Reddit (Hamilton et al., 2017), and graph classification datasets such as MUTAG (Debnath et al., 1991), PROTEINS (Borgwardt et al., 2005), IMDB-B, IMDB-M, COLLAB (Yanardag & Vishwanathan, 2015), etc. Datasets from Open Graph Benchmark (Hu et al., 2020) are also supported. Table A.1 summarizes the statistics of the supported datasets.

B Auto Feature Engineering

We summarize the supported generators in Table B.2, including Graphlets (Milo et al., 2002), EigenGNN (Zhang et al., 2020b), PageRank (Brin & Page, 1998), local degree profile, normalization, one-hot degrees, and one-hot node IDs. For selectors, GBDT (Ke et al., 2017) and Filter-Constant are supported. An automated feature engineering method DeepGL (Rossi et al., 2018) is

Table A.1: The statistics of the supported datasets. For datasets with more than one graph, #Nodes and #Edges are the average numbers of all the graphs. #Features correspond to node features by default, and edge features are specified.

Dataset	Task	#Graphs	#Nodes	#Edges	#Features	#Classes
Cora	Node	1	2,708	5,429	1,433	7
CiteSeer	Node	1	3,327	4,732	3,703	6
PubMed	Node	1	19,717	44,338	500	3
Reddit	Node	1	232,965	11,606,919	602	41
Amazon Computers	Node	1	13,381	245,778	767	10
Amazon Photo	Node	1	7,487	119,043	745	8
Coauthor CS	Node	1	18,333	81,894	6,805	15
Coauthor Physics	Node	1	34,493	247,962	8,415	5
ogbn-products	Node	1	2,449,029	61,859,140	100	47
ogbn-proteins	Node	1	132,534	39,561,252	8(edge)	112
ogbn-arxiv	Node	1	169,343	1,166,243	128	40
ogbn-papers100M	Node	1	111,059,956	1,615,685,872	128	172
Mutag	Graph	188	17.9	19.8	7	2
PTC	Graph	344	14.3	14.7	18	2
ENZYMES	Graph	600	32.6	62.1	3	6
PROTEINS	Graph	1,113	39.1	72.8	3	2
NCI1	Graph	4,110	29.8	32.3	37	2
COLLAB	Graph	5,000	74.5	2,457.8	-	3
IMDB-B	Graph	1,000	19.8	96.5	-	2
IMDB-M	Graph	1,500	13.0	65.9	-	3
REDDIT-B	Graph	2,000	429.6	497.8	-	2
REDDIT-MULTI5K	Graph	5,000	508.5	594.9	-	5
REDDIT-MULTI12K	Graph	11,929	391.4	456.9	-	11
ogbg-molhiv	Graph	41,127	25.5	27.5	9, 3(edge)	2
ogbg-molpcba	Graph	437,929	26.0	28.1	9, 3(edge)	128
ogbg-ppa	Graph	158,100	243.4	2,266.1	7(edge)	37

Table B.2: Supported generators in the auto feature engineering module.

Name	Description
`graphlet`	Local graphlet numbers
`eigen`	EigenGNN features.
`pagerank`	PageRank scores.
`PYGLocalDegreeProfile`	Local Degree Profile features
`PYGNormalizeFeatures`	Row-normalize all node features
`PYGOneHotDegree`	One-hot encoding of node degrees.
`onehot`	One-hot encoding of node IDs

also supported, functioning as both a generator and a selector. For graph feature, Netlsd (Tsitsulin et al., 2018) and a set of graph feature extractors implemented in NetworkX (Hagberg et al., 2008) are wrapped, e.g., `NxTransitivity`, `NxAverageClustering`, etc.