OpenHGNN: An Open-Source Toolkit for Heterogeneous Graph Neural Networks

Hui Han
Beijing University of Posts and Telecommunications
Beijing, China
hanhui@bupt.edu.cn

Tianyu Zhao
Beijing University of Posts and Telecommunications
Beijing, China
tyzhao@bupt.edu.cn

Cheng Yang
Beijing University of Posts and Telecommunications
Beijing, China
Peng Cheng Laboratory
Shenzhen, China
yangcheng@bupt.edu.cn

Hongyi Zhang
Beijing University of Posts and Telecommunications
Beijing, China
hongyizhang@bupt.edu.cn

Yaoqi Liu
Beijing University of Posts and Telecommunications
Beijing, China
yaoqiliu@bupt.edu.cn

Xiao Wang
Beijing University of Posts and Telecommunications
Beijing, China
Peng Cheng Laboratory
Shenzhen, China
xiaowang@bupt.edu.cn

Chuan Shi*
Beijing University of Posts and Telecommunications
Beijing, China
Peng Cheng Laboratory
Shenzhen, China
shichuan@bupt.edu.cn

ABSTRACT

Heterogeneous Graph Neural Networks (HGNNs), as a kind of powerful graph representation learning methods on heterogeneous graphs, have attracted increasing attention of many researchers. Although, several existing libraries have supported HGNNs, they just provide the most basic models and operators. Building and benchmarking various downstream tasks on HGNNs is still painful and time consuming with them. In this paper, we will introduce OpenHGNN, an open-source toolkit for HGNNs. OpenHGNN defines a unified and standard pipeline for training and testing, which can allow users to run a model on a specific dataset with just one command line. OpenHGNN has integrated 20+ mainstream HGNNs and 20+ heterogeneous graph datasets, which can be used for various advanced tasks, such as node classification, link prediction, and recommendation. In addition, thanks to the modularized design of OpenHGNN, it can be extended to meet users' customized needs. We also release several novel and useful tools and features, including leaderboard, autoML, design space, and visualization, to provide users with better usage experiences. OpenHGNN is an open-source project, and the source code is available at https://github.com/BUPT-GAMMA/OpenHGNN.

*The corresponding author.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557664

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Heterogeneous Graph Neural Networks, Heterogeneous Graph Representation Learning, Frameworks

ACM Reference Format:
Hui Han, Tianyu Zhao, Cheng Yang, Hongyi Zhang, Yaoqi Liu, Xiao Wang, and Chuan Shi. 2022. OpenHGNN: An Open-Source Toolkit for Heterogeneous Graph Neural Networks. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557664

1 INTRODUCTION

Heterogeneous graph is a powerful data structure that can be applied in various real-world applications [10, 17, 21]. For example, the e-commerce network, citation network, and social network [15], etc, can be naturally modeled as heterogeneous graphs. In recent years, Heterogeneous Graph Neural Networks (HGNNs), a

Table 1: Common existing open-source GNN and HGNN toolkits.

	Unified Trainer	Heterogeneous Graph	Benchmark	Design Space	HyperParameters Optimization	Visualization
AutoGL [7]	✓	-	-	-	✓	-
GraphGym [22]	✓	✓	-	✓	-	-
OpenKE [8]	-	-	-	-	-	-
OpenAttHetRL [3]	✓	✓	-	-	-	-
CogDL [2]	✓	✓	✓	-	✓	-
HAO Unity [11]	-	✓	-	-	-	✓
TFG [9]	-	-	-	-	-	-
OpenHINE[2]	✓	✓	-	-	-	-
Our OpenHGNN	✓	✓	✓	✓	✓	✓

powerful deep learning method to learn advanced latent representations of heterogeneous graphs, have made remarkable progress. HGNNs can utilize complex heterogeneous graph structure and rich semantic information, which have been widely applied to various downstream tasks such as classification [6, 18, 19], link prediction [13, 20, 23], and recommendation [4, 14, 24].

Despite the great success of HGNNs, there still lack an effective toolkit to support the implementation, deployment, and evaluation of these HGNNs. On one hand, the heterogeneous graphs, with heterogeneity and rich attributes, are usually collected from different sources, resulting in different storage formats. People have to put a lot of work on preprocessing data. On the other hand, most of the HGNNs are sophisticatedly designed and the complex heterogeneity causes very diverse HGNN architectures. Some existing toolkits and libraries, such as CogDL [2], PyG [5], TFG [9], have been developed to help the users efficiently build and train Graph Neural Networks (GNNs). We summarize the characteristics of mainstream toolkits and libraries in Table 1. Most of them concentrate on homogeneous graphs and each one has different characteristics. But they only provide some specific features. For example, GraphGym [22] only provides design space. It remains a major inconvenience for researchers to implement algorithms and empirically compare to baselines, and for beginners to quickly get started learning HGNNs. There is urgent need for an easy-to-use toolkit designed for HGNNs, which can provide standardized and modularized training pipeline and meet the needs of both beginners and advanced users.

Here, we are going to introduce OpenHGNN, an open-source toolkit for HGNNs based on DGL [16] and PyTorch [12]. Our project has gained 250+ stars on GitHub, and all the resources can be found on the official website. OpenHGNN is specifically designed for heterogeneous graph representation learning, integrating many popular HGNNs and datasets. The goal of OpenHGNN is to build a flexible and easy-to-use platform for HGNNs. With OpenHGNN, it is effortless for researchers and engineers to train HGNNs, and also for beginners to quickly learn to go deep into the field. OpenHGNN adopts a modularized design, which means that it has excellent extensibility. Users can develop based on OpenHGNN and modify its sub-modules to meet their needs. With detailed documents and demonstration examples, users can easily define a new model and

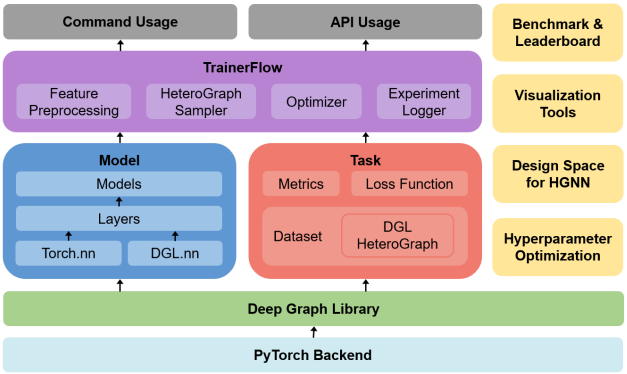

Figure 1: OpenHGNN Architecture

downstream tasks accordingly, such as coming up with a novel loss function or changing the original dataset splitting criterion.

We summarize the characteristics of OpenHGNN as follows:

- **Easy-to-Use**: OpenHGNN provides easy-to-use interfaces for running experiments. Users can start training a model on a specific dataset with just one line command or a several lines python script.
- **Extensibility**: OpenHGNN is modularizedly designed, and users can define customized task/model/dataset to apply a model to specific scenarios.
- **Rich-Functional**: It also integrates several useful tools to help users have a more general and deeper insight into HGNNs.
- **Well-Orginazed Documents**: We provide readable and thorough documents[1] about OpenHGNN, both for the beginners and the proficient users.

In the subsequent sections, we describe each characteristic of OpenHGNN in detail. The framework of OpenHGNN will be introduced in Section 2. Section 3 describes several novel features of OpenHGNN. In Section 4 we will give some examples of how to use and develop based on OpenHGNN. Finally, Section 5 presents the conclusions and our future work.

[1]https://openhgnn.readthedocs.io
[2]https://github.com/BUPT-GAMMA/OpenHINE

2 OPENHGNN FRAMEWORK

In this section, we present the design of OpenHGNN, as shown in Fig.1. Our framework is built on PyTorch [27] and DGL, which are the most popular deep learning libraries for graphs. We abstract three main modules *trainerflow*, *model*, and *task* in OpenHGNN. The *trainerflow*, based on *model* and *task*, unifies and performs the training procedures for HGNNs. We will explain each module in the following part of this section.

2.1 TrainerFlow

In OpenHGNN, *trainerflow* is an abstraction of a predesigned workflow that trains and evaluates a model on a given task. It can be very flexible to cover arbitrary HGNNs training settings. The *trainerflow* consists of two main components, which are named *model* and *task*. HGNNs in OpenHGNN are usually build base on PyTorch and DGL. They will perform forward propagation and output the nodes embeddings, or probability distributions that will be used to calculate the loss. *Task* is an abstraction of loss function, metric and dataset as mentioned above. The two components will be described in detail later. Based on the design of the unified trainerflow and decoupled modules, we could do arbitrary combinations of models, datasets, optimizers, loss functions, etc. For example, if we want to apply HAN [19] to a new dataset, all we need to do is just assigned a new dataset name that OpenHGNN has supported. If we want to design a new HGNN or loss function, we just need to write the core DGL-style codes and OpenHGNN will automatically call it in the predesigned pipeline. Then OpenHGNN will automatically perform training and testing with s few simple settings of trainerflow.

There also are some other necessary components, such as feature preprocessing, heterogeneous graph sampler, optimizer, and experiments logger. Heterogeneous graph datasets, unlike homogeneous ones, usually contain feature loss nodes, making HGNNs cannot be directly applied to this kind of dataset. OpenHGNN can help users solve this problem by generating node embedding in the feature preprocessing step. The HeteroGraphSampler as well as mini-batch training procedures in trainerflow makes HGNNs scalable to datasets that could not be loaded into memory. The logger module is also important for OpenHGNN, and users usually need to track the training history and analyze the performance to get a comprehensive understanding of HGNNs. With the unified trainerflow, OpenHGNN allows native support for users to leverage these tools without much effort.

2.2 Task

Datasets, loss functions, and evaluation metrics are usually bound to each other and closely related to downstream tasks for HGNNs. So as mentioned above, we abstracted *task* as the module to encapsulate these components as well as some extra information. Currently, we support three downstream tasks, namely node classification, link prediction, and recommendation, which are the most widely applied scenarios for HGNNs.

The loss function is necessary for any deep learning model and has been well supported by PyTorch. It will not be described in detail here. There are more or fewer differences in evaluation metrics used by different HGNNs to demonstrate the superior ability in a specific aspect. However, neither DGL nor PyTorch offers standard evaluation metrics, which brings inconvenience to users. OpenHGNN provides a variety of common evaluation metrics, including score of accuracy, recall, MRR, etc. The Dataset instance will automatically download, read to memory, and preprocess to a DGLHeteroGraph instance support the overall training process. It not only maintains input data but also train/test/evaluate split, label information, etc. In OpenHGNN, we have collected and integrated 20+ heterogeneous graph datasets that are frequently used in research area, which meet the common needs of most users. Apart from built-in datasets, OpenHGNN also supports users to customize their datasets by simply subclassing built-in abstract base dataset classes with customized dataset APIs. An example can be found in our documents website.

2.3 Model

Model in OpenHGNN plays the role of encoder that output the node embedding for given input heterogeneous graph data. It mainly contains two parts: model builder and forward propagation. In the model builder part, we should initialize the essential components, e.g., creating a HeteroEmbedLayer layer for the featureless nodes. Forward propagation consists of plenty of complex tensor calculations, because of the complex properties of heterogeneous graphs. This can be confusing for unskilled developers.

Therefore, we abstract and define several HGNN layers that can be combined in the model. HGNN layers will provide users with great flexibility and convenience because we don't have to care about implementation details in HGNNs, but only about how to design model architecture. We have implemented and integrated 20+ HGNN models and 10+ HGNN layers in OpenHGNN, and we will continue contributing to it, including the latest published models based on deep learning and classic models, such as heterogeneous network embedding methods.

3 NOVEL FEATURES

OpenHGNN is not just a collection of dozens of models. We also developed several related tools around HGNNs to give users more convenience and have deeper insights into HGNNs.

3.1 LeaderBoard and BenchMark

We release a leaderboard for popular heterogeneous graph datasets named Open Heterogeneous Graph Benchmark (OHGB[3]) for link prediction and node classification task. In node classification task we release 4 datasets that are `ohgbn-acm`, `ohgbn-imdb` `ohgbn-yelp2`, and `ohgbn-Freebase`. We choose 10 baselines models, and evaluate their score of Macro-F1 and Micro-F1 on these datasets. In link prediction task we release 2 datasets that are `ohgbl-MTWM` and `ohgbl-yelp1`, and the metric we applied is roc-auc score.

3.2 Auto Machine Learning

Hyperparameters optimization (HPO) is an important feature for HGNNs toolkit, since most HGNNs, with more complicated design, utilize more hyperparameters than usual GNNs in general. We integrate a popular library, Optuna [1] into OpenHGNN to enable HPO. Optuna is an automatic hyperparameters optimization

[3]https://github.com/BUPT-GAMMA/OpenHGNN/blob/main/openhgnn/dataset/ohgb.md

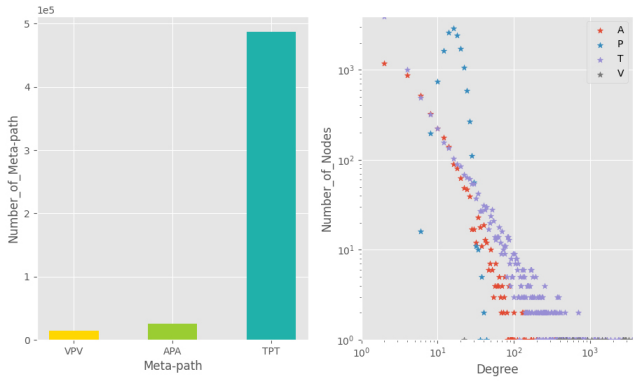

Figure 2: Metapath statistics (left) and degree distribution (right) of DBLP dataset

software framework, particularly designed for machine learning. In OpenHGNN, we have implemented hyperparameters search based on it and user can start a trail in just a few steps. The key of the use HPO is to define a search space, which is the range of each hyperparameter that we should declare in python script `./openhgnn/auto/hpo_space.py`. After defining the search space, all we need to do is just to set `–use_hpo` in the command line. Then the search space will be automatically utilized by OpenHGNN to start searching and output the best results. For specific usage, we put it in Section 4.

3.3 Design Space for HGNNs

Following GraphGym [22], we release a platform Space4HGNN [25] for designing and evaluating HGNNs. We believe Space4HGNN can significantly facilitate the research field of HGNNs. It offers a standardized evaluation pipeline for HGNNs, much like GraphGym for homogeneous GNNs. We also offer parallel launching for faster experiments. Its highlights are summarized below.

3.3.1 Modularized HGNN Implementation. Space4HGNN is easily extendable, allowing future developers to plugin more choices of design dimensions (e.g., a new graph convolution layer). Additionally, it is easy to import new design dimensions to Space4HGNN, such as score function in link prediction.

3.3.2 Standardized HGNN Evaluation. Space4HGNN offers a standardized evaluation pipeline for diverse architecture designs and HGNNs. Benefiting from OpenHGNN, we can evaluate diverse datasets in different tasks easily.

3.4 Visualization Tools

We offer some tools to help users visualize the features of the heterogeneous graphs, e.g., degree distribution and mete-path statistics. These tools allow users to analyze the basic properties of the dataset and fine-tune the model accordingly. Here we give an example of DBLP in Fig.2, which contains 4 node types and 6 edge types, and the total number of nodes and edges are 26,128 and 239,566 respectively.

4 OPENHGNN IN PRACTICE

In this section, we will introduce the basic usage of OpenHGNN and we will give some examples.

OpenHGNN is convenient for researchers to test model performances as baselines in one command. Here is an example:

```
python main.py -m HAN -d acm4HAN -t node_classification
```

Users can choose dataset, model, and task through parameters setting, and OpenHGNN will automatically load dataset and perform model training and evaluation. Furthermore, with the integrated library, Optuna, by defining the search space of hyperparameters in `./openhgnn/auto/hpo_space.py` and passing the parameter `--use_hpo`, OpenHGNN will start to searching and output the best results. OpenHGNN also offers the best hyperparameters and the pretrained model checkpoint for most of the models, and we can use it just by passing `--use_best_config` and `--load_from_pretrained` to the main function.

In addition to supporting command line training, it provides python APIs for user programming. We can set dataset, model, and hyperparameters with the python APIs, which call the low-level OpenHGNN APIs (e.g., trainerflow).

```
import openhgnn
dataset = openhgnn.build_dataset(  # build dataset
    task='node_calssification',
    dataset_name='HGBn-acm')
model = openhgnn.build_model(      # build model
    model_name='HAN')
trainerflow = openhgnn.build_flow( # build trainerflow
    config='conf.yaml',
    model=model,
    dataset=dataset)
trainerflow.train()                # start training
```

OpenHGNN also support customized usage by creating new tasks and developing new models based on existing framework and APIs, which provides users with good extensibility. Due to space limitations, for more detail usages please refer to our documents.

5 CONCLUSION AND FUTURE WORK

This paper introduces an open-source toolkit OpenHGNN, which is extensive, flexible, and easy-to-use. We define a trainerflow for OpenHGNN, a general pipeline that trains and evaluates a model on several tasks, including node classification, link prediction, and recommendation. Users can experiment with mainstream HGNNs and heterogeneous graph datasets integrated in OpenHGNN with just one line of command. We also provide a reproducible leaderboard, as well as useful tools for hyperparameters optimization, design space, and visualization, which can help users understand and explore more characteristics of HGNN in many aspects.In the future, more models for HGNN and datasets will be continuously integrated into OpenHGNN. Besides, we will release more tools and features for HGNNs.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (No. U20B2045, 62192784, 62002029, 62172052, 61772082)

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2623–2631.

[2] Yukuo Cen, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Peng Zhang, Guohao Dai, Yu Wang, Chang Zhou, Hongxia Yang, and Jie Tang. 2021. CogDL: Toolkit for Deep Learning on Graphs. *arXiv preprint arXiv:2103.00959* (2021).

[3] Roohollah Etemadi, Morteza Zihayat, Kuan Feng, Jason Adelman, and Ebrahim Bagheri. 2021. OpenAttHetRL: An Open Source Toolkit for Attributed Heterogeneous Network Representation Learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4706–4710.

[4] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. 2019. Metapath-guided heterogeneous graph neural network for intent recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2478–2486.

[5] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[6] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*. 2331–2341.

[7] Chaoyu Guan, Ziwei Zhang, Haoyang Li, Heng Chang, Zeyang Zhang, Yijian Qin, Jiyan Jiang, Xin Wang, and Wenwu Zhu. 2021. Autogl: A library for automated graph learning. *arXiv preprint arXiv:2104.04987* (2021).

[8] Xu Han, Shulin Cao, Xin Lv, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. Openke: An open toolkit for knowledge embedding. In *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*. 139–144.

[9] Jun Hu, Shengsheng Qian, Quan Fang, Youze Wang, Quan Zhao, Huaiwen Zhang, and Changsheng Xu. 2021. Efficient graph deep learning in tensorflow with tf_geometric. In *Proceedings of the 29th ACM International Conference on Multimedia*. 3775–3778.

[10] Yugang Ji, Chuan Shi, and Xiao Wang. 2021. Prohibited Item Detection on Heterogeneous Risk Graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3867–3877.

[11] Fei Jie, Yanxiang Huang, Qiangwei Bai, and Xindong Wu. 2021. HAO Unity: A Graph-based System for Unifying Heterogeneous Data. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4725–4729.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[13] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[14] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S Yu Philip. 2018. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2018), 357–370.

[15] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 17–37.

[16] 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. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[17] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S Yu Philip. 2022. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data* (2022).

[18] Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. 2021. Self-supervised Heterogeneous Graph Neural Network with Co-contrastive Learning. *knowledge discovery and data mining* (2021).

[19] Wang Xiao, Ji Houye, Shi Chuan, Wang Bai, Cui Peng, Yu P., and Ye Yanfang. 2019. Heterogeneous Graph Attention Network. *WWW* (2019).

[20] Siyong Xu, Cheng Yang, Chuan Shi, Yuan Fang, Yuxin Guo, Tianchi Yang, Luhao Zhang, and Maodi Hu. 2021. Topic-aware Heterogeneous Graph Neural Network for Link Prediction. *conference on information and knowledge management* (2021).

[21] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[22] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 17009–17021.

[23] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.

[24] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 635–644.

[25] Tianyu Zhao, Cheng Yang, Yibo Li, Quan Gan, Zhenyi Wang, Fengqi Liang, Huan Zhao, Yingxia Shao, Xiao Wang, and Chuan Shi. 2022. Space4HGNN: A Novel, Modularized and Reproducible Platform to Evaluate Heterogeneous Graph Neural Network. In *SIGIR*.