PyDHNet: A Python Library for Dynamic Heterogeneous Network Representation Learning and Evaluation

Hoang Nguyen
hoang.cam.nguyen@ryerson.ca
Toronto Metropolitan University, ON, Canada

Radin Hamidi Rad
radin@ryerson.ca
Toronto Metropolitan University, ON, Canada

Ebrahim Bagheri
bagheri@ryerson.ca
Toronto Metropolitan University, ON, Canada

ABSTRACT

Network representation learning and its applications have received increasing attention. Due to their various application areas, many research groups have developed a diverse range of software tools and techniques to learn representation for different types of networks. However, to the best of our knowledge, there are limited works that support representation learning for *dynamic heterogeneous* networks. The work presented in this demonstration paper attempts to fill the gap in this space by developing and publicly releasing an open-source Python library known as, `PyDHNet`, a **Py**thon Library for **D**ynamic **H**eterogeneous **Net**work Representation Learning and Evaluation. `PyDHNet` consists of two main components: dynamic heterogeneous network representation learning and task-specific evaluation. In our paper, we demonstrate that `PyDHNet` has an extensible architecture, is easy to install (through PIP) and use, and integrates quite seamlessly with other Python libraries. We also show that the implementation for `PyDHNet` is efficient and enjoys a competitive execution time.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; • **Mathematics of computing** → *Graph algorithms*; • **Information systems** → *Retrieval models and ranking*.

KEYWORDS

Network Representation Learning, Dynamic Heterogeneous Network, Network Application

ACM Reference Format:
Hoang Nguyen, Radin Hamidi Rad, and Ebrahim Bagheri. 2022. PyDHNet: A Python Library for Dynamic Heterogeneous Network Representation Learning and Evaluation. 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/10.1145/3511808.3557181

1 INTRODUCTION

Network representation learning is the task of mapping a graphical network structure into a lower dimensional embedding vector space. The embedded vectors should be compact to practically efficient while still capturing and maintaining network properties for various downstream network tasks. With the increasing importance of the network representation learning task, several open-source libraries and tools have been progressively developed and shared in the community [7, 11, 12, 17, 22, 25]. We summarize some of the main libraries for this task in Table 1. As shown in the table, the majority of existing libraries are designed for learning network representation for *static* networks. In other words, they consider the graph to be a single snapshot and do not support for evolving graphs over time. Furthermore, these libraries often overlook the need for more complex network representations by only allowing for homogeneous node and edge types in the network.

From the existing libraries, DynamicGEM and CTGCN are among the first to allow representations to be learnt for dynamically evolving graphs. However, both of these libraries are limited by only supporting homogeneous networks. In contrast, OpenAttHetRL and Space4HGNN are the only two libraries that support for network heterogeneity; however, they do not support for network dynamism. The work in this paper illustrates our work on an open-source python-based network representation library, called `PyDHNet`, which supports both network heterogeneity and network dynamism. Mentioned features are not currently supported by any other available libraries. We believe that `PyDHNet` is an extremely valuable contribution to the community in many application domains, such as social networks, biological dataset and traffic monitoring data, to name just a few, deal with graphs that rely on efficient representations through dynamic heterogeneous graphs.

The purpose of this demonstration can be enumerated as follows:

(1) we will introduce `PyDHNet`, which allows users to both efficiently and effectively learn network representations for dynamic heterogeneous networks;
(2) we will show how the structure of `PyDHNet` has been designed based on software engineering principles make it easily extensible for adding new features. Our developed software library is modularized and hence easy to maintain and work with;
(3) we will demonstrate the `PyDHNet` consists of a standalone evaluation library that supports for the testing the performance of the learnt representations on different downstream tasks such as node classification and link prediction, which makes it ideal for comparative analysis and replicable research.

The rest of this demonstration paper is organized as follows: We will first provide an overview of our representation learning technique and how it differentiates itself from other existing work in Section 2. Next, the software architecture of `PyDHNet` is briefly

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '22, October 17–21, 2022, Atlanta, GA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557181

Tool	**First release**	**Library Features**				**Link**
		Environment	**Heterogeneity**	**Node properties**	**Evaluation**	
GEM [10]	2017	Static	Homogeneous	No	No	https://github.com/palash1992/GEM
OpenNE [20]	2018	Static	Both	No	No	https://github.com/thunlp/OpenNE
DynamicGEM [9]	2018	Dynamic	Homogeneous	No	No	https://github.com/palash1992/DynamicGEM
OpenHINE [27]	2020	Static	Heterogeneous	No	No	https://github.com/BUPT-GAMMA/OpenHINE
OpenAttHetRL [3]	2021	Static	Heterogeneous	Yes	No	https://github.com/etemadir/OpenAttHetRL
CTGCN [15]	2021	Both	Homogeneous	No	No	https://github.com/jhljx/CTGCN
Space4HGNN [27]	2022	Static	Heterogeneous	Yes	No	https://github.com/BUPT-GAMMA/OpenHGNN
PyDHNet	**2022**	**Dynamic**	**Heterogeneous**	**No**	**Yes**	**https://github.com/hoangntc/PyDHNet**

Table 1: The key features of some common open-source tools for network representation learning.

introduced in Section 3. We will then show in Section 4 that PyDHNet has an easy-to-use API interface that makes it easy to utilize in any Python program. We compare PyDHNet and its competitors from execution time perspective in Section 5. Finally, Section 6 presents final concluding remarks.

2 DYNAMIC HETEROGENEOUS NETWORK REPRESENTATION LEARNING

In general, the dynamic embedding of a node in a dynamic network can be viewed as a time series of vectors where each single vector is associated with a timestamp in the network. Equivalently, one can consider a dynamic network to be a series of static network snapshots, which can be denoted as $\mathcal{G} = \left\{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^T\right\}$ where T is the number of timestamps. A static network at time t is defined as $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$, where $\mathcal{V}^t$ is the node set and $\mathcal{E}^t$ is the edge set. In order to deal with heterogeneous networks, at each timestamp t, we formalize $\mathcal{G}^t$ as a static heterogeneous network, which consists of multiple node types. As such and in each timestamp t, our dynamic network representation learning method aims to learn latent representations for each node in the network $\mathcal{G}$ in such a way that it preserves both the properties of the node in previous and current heterogeneous network snapshots as well as its evolutionary behavior up to time t.

There are two main approaches to deal with heterogeneity at each network snapshot: (1) one approach adopts a type-specific mapping function to aggregate the information from the directed neighbors (e.g., HGAT [22], HGT [11]); (2) the other approach captures higher-order proximity by employing meta-path neighborhood aggregation (e.g., MAGNN [7], HeteGNNs [12]). The implementation of such methods can be found in the latest libraries such as OpenNE and Space4HGNN. However, none of these libraries consider the subgraph structure of the node when learning network representations. The network learning method used in our library utilizes a meta-path neighborhood aggregation approach to capture higher-order proximity information while at the same time considering the subgraph structures around each node. As such, our approach for capturing heterogeneity is unique as it captures both local subgraph structures surrounding each node as well as higher-order proximity information.

For the sake of encoding the temporal nature of the dynamic graph, a natural approach is to use time series encoders since they support node addition/deletion and link update [19]. However, the research community has recently found that the use of positional encodings is a more effective approach compared to recurrent neural networks especially due to its computational efficiency [1, 18]. This approach allows modelling temporal dependencies without the need of feeding the sequential data recursively, which supports parallel computation. Our library uses Transformer-like positional encoding [8] in order to capture the temporal order of network snapshots. In our library, the positional encoding first encodes the order of the network snapshots regarding the position of their timestamp in the whole time series of the network. The position encoding vectors are then combined with the input vectors to model the sequence of the input representation.

Based on these techniques, our representation learning technique can capture both the heterogeneity well as the evolution of dynamic heterogeneous networks, and allows users to have richer network representations with potentially better performance on downstream tasks.

3 PYDHNET STRUCTURE

In this section, we present the design of the PyDHNet library and show that is quite easy to use and extend it by other users. The overall structure of PyDHNet is shown in Figure 1. As shown in this figure, the library consists of two main components, *Network Representation Learning* designed for developing embedding representation, and *Network Evaluation* for assessing the quality of the learnt representations on downstream tasks. The former component consists of three main tasks, namely (1) data preparation, (2) model training, and (3) embedding generation. The latter component supports the evaluation of the embedding representations based on downstream tasks such as node classification and link prediction. This would make PyDHNet quite suitable to serve as baseline for many domain-specific tasks such as expertise prediction (node classification) [13, 21], and product recommendation (link prediction) [2, 5], to name a few.

The network representation learning component of the PyDHNet library is built over PyTorch Geometric (PyG) [6] and Pytorch Lightning [4] while the network evaluation component is built to be compatible with Scikit-learn [16]. The mentioned libraries are widely used in the Machine Learning (ML) community and have well-adopted standards for data input, model export and evaluation. As a result, PyDHNet offers the following very desirable characteristics: (1) it is quite easy to use as it can be installed simply through the 'PIP' command; (2) it relies on dependencies that are already well-managed through the Pytorch libraries; and is easy to learn as it uses Pytorch standards for development and Scikit-learn standards for model training, export and evaluation.

3.1 Network Representation Learning Component

We have implemented the network representation learning component of PyDHNet in an object-oriented programming (OOP) style with the back-end based on well-known set of Python libraries, i.e.,

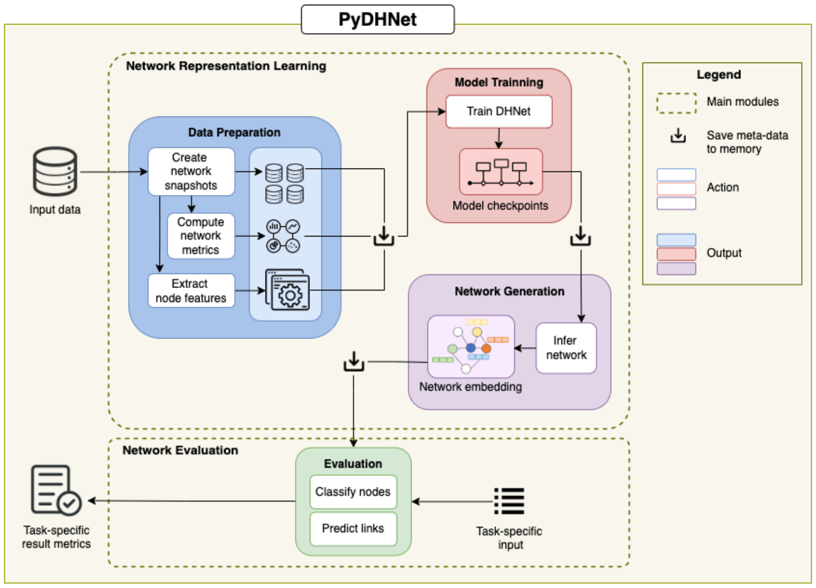

Figure 1: The overall structure of PyDHNet.

Pytorch, Pytorch Lightning and Pytorch Geometric. As a result, this allows users in this space to easily pick up our library and use it without much learning barriers for their specific downstream task.
Data Preparation In this task, the users are able to convert the format of the input dataset to the general standard format used for graph. We support a similar graph structure input to PyG except that our graph snapshots require temporal labels as well to support for dynamicity. We note that given the fact that we deal with dynamic networks, each graph consists of multiple snapshots. Each network snapshot is considered to be a static heterogeneous network and wrapped up as `Dataset` object from PyG. The network features at the node level are then extracted using the PyG library which enables the standardization of the dimension of node features used in the later stages.
Model Training In the model training task, `PyDHNet` provides support for data loading, model training and model evaluation. Three sub-modules, namely `DataModule`, `ModelModule` and `Trainer`, are responsible for the mentioned tasks, respectively. The `DataModule` handles the construction and the flow of the large network data during training process. In order to support for the streaming nature of dynamic networks, this module loads the data in mini batches with the temporal and structural information related to a node such as the node features, the node type and the structural subgraphs it belongs to in each timestamp. The `ModelModule` handles the model generation task and receives the output of the `DataModule` as its input and generates the appropriate embeddings for the target network. The optimization during the learning process is defined under the `Trainer`. The `Trainer` provides commonly-used utility functions such as early stopping, regularization and checkpoint tracking. The configuration of all of the mentioned sub-modules such as the data batch size, the learning rate, the dropout probability or the loss criteria are stored in a `.json` file or can be inputted via a dictionary. These sub-modules are inherited from the `LightningDataModule`, `LightningModule` and `Trainer` classes of the Pytorch Lightning library, which allows the users to easily and quickly implement or customize their own learning process.
Network Generation The objective of this module is to generate the embedding vectors for all the nodes in the network that can later be used for downstream tasks, e.g., node classification or link prediction. We note that `PyDHNet` generates node representations for each node up to each timestamp that it has observed. Therefore, the node representation will be updated at each timestamp.

3.2 Network Evaluation Component

Given a specific task, the network evaluation component receives the embedding vectors of all nodes or a set of node pairs as the input and employs a machine learning predictor (e.g., Logistic Regression) or a statistical measures (e.g., sigmoid function) to predict the score of the targeted output. Finally, it calculates a set of widely-used evaluation metrics, including but not limited to, accuracy, recall and precision, based on comparison between the ground truth and the predicted results.

4 PYDHNET DEMONSTRATION

In this section, we practically demonstrate how `PyDHNet` can be easily integrated into any Python program using its easy to use API. The `PyDHNet` tool requires four input files:

- **node_types** gives the detail of the type of all nodes in the network.
- **temporal_edge_list** tracks the existence of the edges corresponding to different timestamps.
- **temporal_subgraphs** includes the information of all the structural subgraphs in the network.
- **data** stores the information related to the nodes including the node index, the subgraph indices which it belongs to and whether the node is used for training/validation/testing.

To sample the structural subgraphs corresponding to each node at each time step, we provide a `TemporalSubgraphSampler`, which provides the implementation for some built-in sampling functions such as `DiffusionSampler` and `RandomWalkSampler`. An example on how to use the subgraph sampler is depicted in Figure 2.

```python
from PyDHNet.subgraph_sampler import TemporalSubgraphSampler

sampler = TemporalSubgraphSampler(
    node_path,
    edge_path,
    sampled_node_ids,
    max_size=5,
    number_of_nodes=20,
    seed=0,
    output_dir='./',
)

sampler.sampling_temporal_subgraph()
sampler.write_temporal_subgraphs()
```

Figure 2: Code Snippet for Temporal Subgraph Sampler.

Figure 3 demonstrates the implementation for the network representation learning component. The main component of `PyDHNet` can be initialized either by a file path which points to a configuration file or a direct configuration dictionary. Once the instance is created, the users are able to run the whole pipeline of the process in one full execution or in separate steps, both of which are supported.

```python
from PyDHNet import PyDHNet

# Instance initialization
pydhnet = PyDHNet(config_path='./PyDHNet/config/dblp.json')

# or
config_dict = {
    'name': 'dblp',
    'num_time_steps': 8,
    'batch_size': 64,
    'learning_rate': 0.0001,
    'checkpoint_dir': './model',
}
pydhnet = PyDHNet(config_dict=config_dict)
```

```python
#----------------------------------------
# Manual running
# 1. Data preprocessing
pydhnet.preprocess_data()

# 2. DataModule, ModelModule, Trainer initialization
data_module, model_module, trainer = pydhnet.initialize()
pydhnet.train(data_module, model_module, trainer)

# 3. Embedding generating
restore_model_dir = str(pydhnet.config['checkpoint_dir'])
restore_model_name = 'name.ckpt'
output_dir = str(PROJ_PATH / 'output')
pydhnet.generate_embedding(
    data_module, model_module, restore_model_dir, restore_model_name, output_dir)
```

```python
#----------------------------------------
# Full pipeline running
pydhnet.run_pipeline()
```

Figure 3: Code Snippets for the three tasks offered by the *Network Representation Learning* component.

After obtaining the learnt model and inferring the network embedding, it is possible to easily evaluate the learnt representations using the `PyDHNet` network evaluation component. For clarity, we demonstrate how evaluation based on two downstream tasks can be performed in Figure 4. It is worth noting that evaluation is an independent component of our library, which can be used in conjunction with other network representation libraries as well. The evaluation component only requires the embeddings generated by any network representation learning libraries (often referred to as the node/node pair features), the label (the class of the node w.r.t. the task) and the sample dataset indicator (whether the sample is for training/validation/testing) in order to produce the evaluation metrics.

5 EXECUTION PERFORMANCE

It is also important to comparatively analyze the execution time of `PyDHNet` and the latest graph representation libraries, namely CTGCN and OpenHINE. While other libraries do not simultaneously

```python
from PyDHNet.evaluation import (
    eval_link_prediction,
    eval_node_classification
)

lp_result = eval_link_prediction(
    source_features,
    target_features,
    labels,
    train_val_test_index
)

nc_result = eval_node_classification(
    features,
    labels,
    train_val_test_index
)
```

Figure 4: Code Snippet for Network Evaluation.

support for dynamicity and heterogeneity, we show that `PyDHNet` still shows competitive execution time. In order to compare the performance of different libraries, we run experiments over the DBLP dataset for the node classification task, which aims to predict the research area of an author as suggested in [21, 24]. Based on the dataset, we construct a network with 5,343 papers, 1,475 authors, 1,951 terms and 5 venues forming 55,310 interactions between the nodes in total. There are 8 timestamps in the dataset. For tools that support only static graphs, we consolidate nodes and edges from all timestamps into one graph. For methods that support only homogeneous nodes, we consider all nodes to be of the same type as suggested by [23, 26].

In all the experiments, the hidden dimension is set to 128, the batch size is 64 and the number of iterations is 50. We report execution time on a machine with one NVIDIA GeForce RTX 3090.

Model	Tool	Network	Execution Time
GCN [14]	CTGCN	Static Homogeneous Network	6 mins
MetaGraph2Vec [25]	OpenHINE	Static Heterogeneous Network	35 mins
CTGCN [15]	CTGCN	Dynamic Homogeneous Network	2 hours 25 mins
PyDHNet	PyDHNet	Dynamic Heterogeneous Network	1 hours 45 mins

Table 2: The execution time of some common open-source tools for network representation learning.

The execution times are reported in Table 2 for the different methods. As seen in the table, methods that are designed for homogeneous networks are faster than those for heterogeneous networks (e.g., compare GCN with MetaGraph2Vec), and dynamic networks are lengthier to process compared to static networks (compare CTGCN with MetaGraph2Vec). We note however `PyDHNet` has an efficient implementation where its execution time for a dynamic *'heterogeneous'* network is even faster than other strong implementations for dynamic 'homogeneous' networks.

6 CONCLUDING REMARKS

In this demonstration paper, we have introduced the open-source python-based `PyDHNet` library that supports the task of learning embedding representations for Dynamic Heterogenous Network Representation Learning. We have shown that `PyDHNet` provides support for both dynamic (temporal) and heterogeneous networks, which are currently not supported by existing tools. Furthermore, we explained how `PyDHNet` is well-structured, extensible and easy to use, which makes it suitable for both practical use and further development. In addition, we have empirically shown the the implementation of `PyDHNet` is quite efficient as it even has a faster performance compared to existing state of the art dynamic *homogeneous* network representation learning tools.

REFERENCES

[1] Ling Cai, Krzysztof Janowicz, Gengchen Mai, Bo Yan, and Rui Zhu. 2020. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Trans. GIS* 24, 3 (2020), 736–755. https://doi.org/10.1111/tgis.12644

[2] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. 2021. Sequential Recommendation with Graph Neural Networks. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 378–387. https://doi.org/10.1145/3404835.3462968

[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] William Falcon et al. 2019. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019), 6.

[5] Ziwei Fan, Zhiwei Liu, Jiawei Zhang, Yun Xiong, Lei Zheng, and Philip S. Yu. 2021. Continuous-Time Sequential Recommendation with Temporal Graph Collaborative Transformer. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 433–442. https://doi.org/10.1145/3459637.3482242

[6] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* abs/1903.02428 (2019). arXiv:1903.02428 http://arxiv.org/abs/1903.02428

[7] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 2331–2341. https://doi.org/10.1145/3366423.3380297

[8] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, 1243–1252. http://proceedings.mlr.press/v70/gehring17a.html

[9] Palash Goyal, Sujit Rokka Chhetri, Ninareh Mehrabi, Emilio Ferrara, and Arquimedes Canedo. 2018. DynamicGEM: A Library for Dynamic Graph Embedding Methods. *CoRR* abs/1811.10734 (2018). arXiv:1811.10734 http://arxiv.org/abs/1811.10734

[10] Palash Goyal and Emilio Ferrara. 2018. GEM: A Python package for graph embedding methods. *J. Open Source Softw.* 3, 29 (2018), 876. https://doi.org/10.21105/joss.00876

[11] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 2704–2710. https://doi.org/10.1145/3366423.3380027

[12] Houye Ji, Xiao Wang, Chuan Shi, Bai Wang, and Philip Yu. 2021. Heterogeneous Graph Propagation Network. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[13] Di Jin, Cuiying Huo, Chundong Liang, and Liang Yang. 2021. Heterogeneous Graph Neural Network via Attribute Completion. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 391–400. https://doi.org/10.1145/3442381.3449914

[14] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).

[15] Jingxin Liu, Chang Xu, Chang Yin, Weiqiang Wu, and You Song. 2020. K-core based temporal graph convolutional network for dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[17] Radin Hamidi Rad, Aabid Mitha, Hossein Fani, Mehdi Kargar, Jaroslaw Szlichta, and Ebrahim Bagheri. 2021. PyTFL: A Python-based Neural Team Formation Toolkit. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 4716–4720. https://doi.org/10.1145/3459637.3481992

[18] Selim Reza, Marta Campos Ferreira, JJM Machado, and João Manuel RS Tavares. 2022. A multi-head attention-based transformer model for traffic flow forecasting with a comparative analysis to recurrent neural networks. *Expert Systems with Applications* 202 (2022), 117275.

[19] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. 2020. Foundations and modelling of dynamic networks using Dynamic Graph Neural Networks: A survey. *CoRR* abs/2005.07496 (2020). arXiv:2005.07496 https://arxiv.org/abs/2005.07496

[20] Cunchao Tu, Yuan Yao, Zhengyan Zhang, Ganqu Cui, Hao Wang, Changxin Tian, Jie Zhou, and Cheng Yang. 2018. OpenNE: An Open Source Toolkit for Network Embedding.

[21] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019. Heterogeneous Graph Attention Network. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia (Eds.). ACM, 2022–2032. https://doi.org/10.1145/3308558.3313562

[22] Tianchi Yang, Linmei Hu, Chuan Shi, Houye Ji, Xiaoli Li, and Liqiang Nie. 2021. HGAT: Heterogeneous Graph Attention Networks for Semi-supervised Short Text Classification. *ACM Trans. Inf. Syst.* 39, 3 (2021), 32:1–32:29. https://doi.org/10.1145/3450352

[23] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems* 32 (2019).

[24] 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 793–803. https://doi.org/10.1145/3292500.3330961

[25] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. MetaGraph2Vec: Complex Semantic Path Augmented Heterogeneous Network Embedding. In *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part II (Lecture Notes in Computer Science)*, Dinh Q. Phung, Vincent S. Tseng, Geoffrey I. Webb, Bao Ho, Mohadeseh Ganji, and Lida Rashidi (Eds.), Vol. 10938. Springer, 196–208. https://doi.org/10.1007/978-3-319-93037-4_16

[26] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. 2021. Heterogeneous Graph Structure Learning for Graph Neural Networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021.* AAAI Press, 4697–4705. https://ojs.aaai.org/index.php/AAAI/article/view/16600

[27] 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. *CoRR* abs/2202.09177 (2022). arXiv:2202.09177 https://arxiv.org/abs/2202.09177