Journal of Machine Learning Research 23 (2022) 1-48 Submitted 5/22; Revised 11/22; Published 12/22

Benchmarking Graph Neural Networks

Vijay Prakash Dwivedi[1] VIJAYPRA001@E.NTU.EDU.SG

Chaitanya K. Joshi[2] CHAITANYA.JOSHI@CL.CAM.AC.UK

Anh Tuan Luu[1] ANHTUAN.LUU@NTU.EDU.SG

Thomas Laurent[3] TLAURENT@LMU.EDU

Yoshua Bengio[4] YOSHUA.BENGIO@MILA.QUEBEC

Xavier Bresson[5] XAVIERCS@NUS.EDU.SG

[1]*Nanyang Technological University, Singapore,* [2]*University of Cambridge, UK,* [3]*Loyola Marymount University, USA,* [4]*Mila, University of Montréal, Canada,* [5]*National University of Singapore*

Editor: Joaquin Vanschoren

Abstract

In the last few years, graph neural networks (GNNs) have become the standard toolkit for analyzing and learning from data on graphs. This emerging field has witnessed an extensive growth of promising techniques that have been applied with success to computer science, mathematics, biology, physics and chemistry. But for any successful field to become mainstream and reliable, benchmarks must be developed to quantify progress. This led us in March 2020 to release a benchmark framework that i) comprises of a diverse collection of mathematical and real-world graphs, ii) enables fair model comparison with the same parameter budget to identify key architectures, iii) has an open-source, easy-to-use and reproducible code infrastructure, and iv) is flexible for researchers to experiment with new theoretical ideas. As of December 2022, the GitHub repository[1] has reached 2,000 stars and 380 forks, which demonstrates the utility of the proposed open-source framework through the wide usage by the GNN community. In this paper, we present an updated version of our benchmark with a concise presentation of the aforementioned framework characteristics, an additional medium-sized molecular dataset AQSOL, similar to the popular ZINC, but with a real-world measured chemical target, and discuss how this framework can be leveraged to explore new GNN designs and insights. As a proof of value of our benchmark, we study the case of graph positional encoding (PE) in GNNs, which was introduced with this benchmark and has since spurred interest of exploring more powerful PE for Transformers and GNNs in a robust experimental setting.

Keywords: Graph Neural Networks, Benchmarking, Graph Datasets, Exploration Tool

1. Introduction

Graph neural networks have benefitted from a great interest recently with numerous methods being developed for diverse domains including chemistry (Duvenaud et al., 2015; Gilmer et al., 2017), physics (Cranmer et al., 2019; Sanchez-Gonzalez et al., 2020), social sciences (Monti et al., 2019), transportation (Derrow-Pinion et al., 2021), knowledge graphs (Schlichtkrull et al., 2018; Chami et al., 2020), recommendation (Monti et al., 2017b; Ying et al., 2018), and

[1] The framework is hosted at `https://github.com/graphdeeplearning/benchmarking-gnns`.

neuroscience (Griffa et al., 2017). Developing powerful and expressive GNN architectures is a key concern towards practical applications and real-world adoption of graph machine learning. However, tracking progress is often challenging in the absence of a community-standard benchmark as models that are evaluated on traditionally-used datasets with inconsistent experimental comparisons make it difficult to differentiate complex, simple and graph-agnostic architectures (Hoang and Maehara, 2019; Chen et al., 2019a; Errica et al., 2019).

We introduce an open-source GNN benchmarking framework (see Fig 1) that brings forward a set of diverse medium-scale datasets which are discriminative to benchmark different GNN models when compared fairly on fixed parameter budgets. The existing collection of datasets, the protocol to use the same parameter budgets for comparison, and the modular coding infrastructure has been widely used to prototype powerful GNN ideas and develop new insights, as shown by 2000+ stars and 380+ forks of the GitHub repository from its first release in March 2020, and 470+ citations gathered by the ArXiv technical report according to Google Scholar. Aspects of the benchmark have led to facilitating several interesting studies for GNNs such as on (i) the aggregation functions and filters (Corso et al., 2020; Tailor et al., 2021; Elhag et al., 2022), (ii) improving expressive power of GNNs (Valsesia et al., 2021; Bouritsas et al., 2022; Bevilacqua et al., 2021), (iii) pooling mechanisms (Mesquita et al., 2020), (iv) graph-specific normalization and regularization (Chen et al., 2022; Zhou et al., 2020; Zhang et al., 2021), and (v) GNNs' robustness and efficiency (Wei and Hu, 2022; Tailor et al., 2020) among other ideas contributed in the literature. In this paper, we provide an updated overview of the proposed framework that extends on the previous collection of datasets to (a) include a number of essential mathematical datasets which can be used to test specific theoretical graph properties, and (b) incorporate another molecular dataset, AQSOL (Sorkun et al., 2019) that has real-world experimental solubility targets unlike ZINC's computed targets, resulting in a collection of 12 datasets (see Table 1). The remainder of the paper discusses a proof of concept of the benchmark that can be used to explore and develop new insights for GNNs.

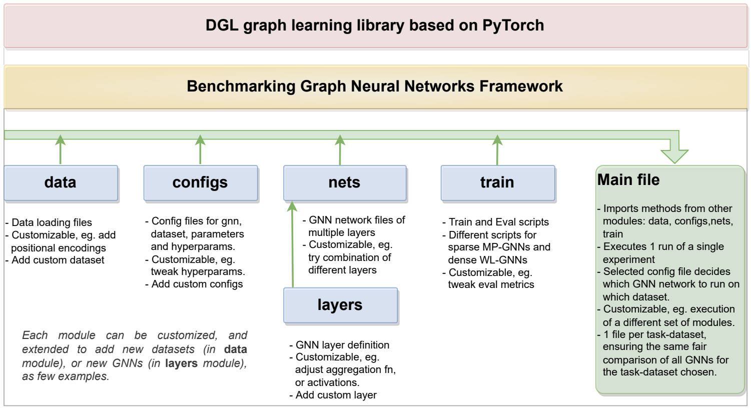

DGL graph learning library based on PyTorch
Benchmarking Graph Neural Networks Framework
data
- Data loading files
- Customizable, eg. add positional encodings
- Add custom dataset
configs
- Config files for gnn, dataset, parameters and hyperparams.
- Customizable, eg. tweak hyperparams.
- Add custom configs
nets
- GNN network files of multiple layers
- Customizable, eg. try combination of different layers
layers
- GNN layer definition
- Customizable, eg. adjust aggregation fn, or activations.
- Add custom layer
train
- Train and Eval scripts
- Different scripts for sparse MP-GNNs and dense WL-GNNs
- Customizable, eg. tweak eval metrics
Main file
- Imports methods from other modules: data, configs,nets, train
- Executes 1 run of a single experiment
- Selected config file decides which GNN network to run on which dataset.
- Customizable, eg. execution of a different set of modules.
- 1 file per task-dataset, ensuring the same fair comparison of all GNNs for the task-dataset chosen.
Each module can be customized, and extended to add new datasets (in data module), or new GNNs (in layers module), as few examples.

Figure 1: Overview sketch of the proposed GNN benchmarking framework with different modular components. This benchmark is built upon DGL and PyTorch libraries.

2. Overview of GNN Benchmarking Framework

Datasets. Collecting representative, realistic and medium-to-large scale graph datasets presents several challenges. It is unclear what theoretical tools can define the quality of a dataset or validate its statistical representativeness for a given task. Similarly, there are several arbitrary choices when preparing graphs, such as node and edge features. Finally, very large graph datasets also present a computational challenge and require extensive GPU resources to be studied (Chiang et al., 2019; Rossi et al., 2020; Hu et al., 2021).

On account of such challenges, we present in our benchmark a collection of 12 graph datasets, listed in Table 1, which are (i) collected from real-world sources and generated from mathematical models, (ii) of medium-scale size suitable for academic research, (iii) representative of the three fundamental learning tasks at graph-level, node-level and edge-level, and (iv) from diverse end-application domains. These datasets are appropriate to statistically separate the performance of GNNs on specific graph properties, hence fulfilling the academic mission to identify first principles.

Domain	Dataset	Task
A. Real World Graphs		
Chemistry	ZINC AQSOL	Graph Regression
Social/Academic Networks	OGBL-COLLAB WikiCS	Edge Classification Node Classification
Computer Vision	MNIST CIFAR10	Graph Classification
B. Mathematical Graphs		
Mathematical Modelling	PATTERN CLUSTER	Node Classification
Combinatorial Optimization	TSP	Edge Classification
Isomorphism	CSL	Graph Classification
Cycles in Graphs	CYCLES	Graph Classification
Multi Graph Properties	GraphTheoryProp	Multi Node/Graph Task

Table 1: Summary statistics of datasets included in the benchmark. Additional details in Appendix Table 2 and Sec. C.

Coding Infrastructure. Our benchmarking infrastructure builds upon PyTorch (Paszke et al., 2019) and DGL (Wang et al., 2019), and has been developed with the following fundamental objectives: (a) Ease-of-use and modularity, enabling new users to experiment and study the building blocks of GNNs; (b) Experimental rigour and fairness for all models being benchmarked; and (c) Being future-proof and comprehensive for tracking the progress of graph ML tasks and new GNNs. At a high level as sketched in Fig 1, our benchmark unifies independent components for: (i) Data pipelines; (ii) GNN layers and models; (iii) Training and evaluation functions; (iv) Network and hyperparameter configurations; and (v) Scripts for reproducibility. This standardized framework has been of immense help to the community as aforementioned about its wide community usage. It has enabled researchers to explore new ideas at any stage of the pipeline without setting up everything else. We direct readers to the `README` user manual included in our GitHub repository for detailed instructions on using the coding infrastructure.

Parameter Budgets for Fair Comparison. One goal of this benchmark is not to find the optimal hyperparameters for a specific model (which is computationally expensive), but to compare the model and their building blocks within a budget of parameters. Therefore, we decide on using two model parameter budgets: i) 100k for each GNN for all the datasets, and ii) 500k for GNNs where the scalability of a model to larger parameters and deeper layers are investigated. The layers and dimensions are selected accordingly to match these budgets.

Discussion on Design Choices. First, our motivation behind the medium-scale datasets in the benchmark is to enable swift yet reliable prototyping of GNN research ideas as we can achieve statistical difference in GNN performance within 12 hours of single experiment runs (see Appendix I). Medium-scale datasets are arguably more informative than small datasets and more feasible than large-scale datasets in the academic-scale research. **Second**, our coding infrastucture with standard protocols has enabled fair comparison of GNNs something that was lacking in prior literature (Errica et al., 2019). **Third**, a fixed budget of model parameters for each GNN model allows for fair comparison of different architectures. In the absence of such design choice, it is comparatively difficult to conclude whether a better performing model's gain arises from its architectural design or extra learning capacity brought by additional model parameters. **Finally**, the aforementioned decisions can be refined and extended to allow further flexibility as elaborated in Appendix H.

3. How can the benchmark be used to explore new insights?

The proposed benchmarking framework can be used to test new research ideas at the level of data preprocessing, improving the GNN layers and normalization schemes, or even to substantiate the performance of a novel GNN model. Such studies are conveniently facilitated given the set of diverse datasets and the rigorous comparison of different experiments on same parameter budgets. At any stage, a modular component of the framework, such as **data**, **layers**, etc., can be modified and multiple experiments on the datasets can be conducted fairly and with ease. Indeed, we employ the framework to perform multiple studies, out of which we present here the insight of positional encodings for GNNs using Laplacian eigenvectors, for an example, while the remainder is included in the appendix.

Graph Positional Encoding. Nodes in a graph do not have any canonical positional information. In the absence of available features, nodes are anonymous, such as the nodes in CSL, CYCLES or GraphTheoryProp datasets in our benchmark. As such, message passing based GCNs perform either poorly or fail completely to detect the class of the graph, such as isomorphic class, or cycles (Murphy et al., 2019; Loukas, 2020). We proposed the use of Laplacian eigenvectors (Belkin and Niyogi, 2003) as node positional encoding by building on top of corresponding dataset files in the **data** module as shown in the pseudo-code snippet alongside. In other words, the positional encoding p_i for a node i can be added to its features x_i as $x_i = x_i + p_i$. Similarly, other ideas can be explored by leveraging respective modules of the framework (in Fig 1) for which we direct to `README` of our GitHub repository.

We used the benchmark to validate and also quantified the improvement provided by this idea. The Laplacian PE effectively improved the MP-GCNs (message-passing based Graph Convolutional Networks) on the the 3 synthetics datasets mentioned previously and other real-world datasets, including the newly added AQSOL dataset. A detailed presentation of the PE with experiments are in Appendix E.1. After the introduction of Laplacian PE through this benchmark, new ideas followed up in the literature for improving PE (Beaini et al., 2021; Wang et al., 2022; Lim et al., 2022; Kreuzer et al., 2021; Ying et al., 2021; Mialon et al., 2021), thus demonstrating how the identification of first principles using the proposed benchmark can steer GNN research.

```
class NameOfDataset(torch.utils.data.Dataset):
    def __init__(self, name=‘name_of_dataset’):
        # existing code to load dataset

    def _add_positional_encodings(self, args):
        # new code that precomputes and adds
        # positional encoding using eigenvectors
```

Figure 2: Primary code block in **data** module to implement Graph PE.

4. Conclusion

This paper introduces an open-source benchmarking framework for Graph Neural Networks that is modular, easy-to-use, and can be leveraged to quickly yet robustly test new GNN ideas and explore insights that direct further research. The benchmark led us to propose graph PE that has remained an interesting avenue of exploration since the first release of our benchmark. We also perform additional studies on investigation of different GNN categories, and edge representations for link prediction, the details of which are included in the appendix for interested readers.

Acknowledgments

XB is supported by NRF Fellowship NRFF2017-10, NUS-R-252-000-B97-133 and A*STAR Grant ID A20H4g2141. This research is supported by Nanyang Technological University, under SUG Grant (020724-00001). The authors thank the reviewers and the editor for their comments and suggestions, which greatly improved the manuscript.

References

Emmanuel Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282, November 2012. ISSN 0162-8828.

David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver, 2006.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *NeurIPS workshop on Deep Learning*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

Dominique Beaini, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. In *International Conference on Machine Learning*, pages 748–758. PMLR, 2021.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910*, 2021.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

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

Xavier Bresson and Thomas Laurent. A two-step graph convolutional decoder for molecule generation. In *NeurIPS Workshop on Machine Learning and the Physical Sciences*, 2019.

Marc Brockschmidt. Gnn-film: Graph neural networks with feature-wise linear modulation. *arXiv preprint arXiv:1906.12192*, 2019.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. Low-dimensional hyperbolic knowledge graph embeddings. *arXiv preprint arXiv:2005.00545*, 2020.

Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification, 2019a.

Yihao Chen, Xin Tang, Xianbiao Qi, Chun-Guang Li, and Rong Xiao. Learning graph normalization for graph neural networks. *Neurocomputing*, 2022.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, pages 15868–15876, 2019b.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

Miles D Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks. *arXiv preprint arXiv:1909.05862*, 2019.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*, pages 3844–3852. 2016.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

Ahmed AA Elhag, Gabriele Corso, Hannes Stärk, and Michael M Bronstein. Graph anisotropic diffusion for molecules. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification, 2019.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nystrom method. *IEEE transactions on pattern analysis and machine intelligence*, 26 (2):214–225, 2004.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

Alessandra Griffa, Benjamin Ricaud, Kirell Benzi, Xavier Bresson, Alessandro Daducci, Pierre Vandergheynst, Jean-Philippe Thiran, and Patric Hagmann. Transient networks of spatio-temporal connectivity map communication pathways in brain functional systems. *NeuroImage*, 155:490–502, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

NT Hoang and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *ArXiv*, abs/1905.09550, 2019.

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. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. Redundancy-free computation graphs for graph neural networks. *arXiv preprint arXiv:1906.03707*, 2019.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

Chaitanya Joshi. Transformers are graph neural networks. *The Gradient*, 2020.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, pages 1–29, 2022.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Boris Knyazev, Graham W Taylor, and Mohamed R Amer. Understanding attention and generalization in graph neural networks. *arXiv preprint arXiv:1905.02850*, 2019.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012.*, pages 1106–1114, 2012.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. 1995.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding–design provably more powerful gnns for structural representation learning. *arXiv preprint arXiv:2009.00142*, 2020.

Derek Lim, Joshua David Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. In *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*, 2022.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=B1l2bp4YwS`.

Enxhell Luzhnica, Ben Day, and Pietro Liò. On graph classification networks, datasets and baselines. *arXiv preprint arXiv:1905.04682*, 2019.

Jitendra Malik. Technical perspective: What led computer vision to deep learning? *Commun. ACM*, 60(6):82–83, May 2017. ISSN 0001-0782.

Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*, 2017.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2153–2164, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *International Conference on Learning Representations*, 2019b.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. *International Conference on Machine Learning*, 2019c.

Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231, 2020.

Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017a. doi: 10.1109/cvpr.2017.576.

Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707, 2017b.

Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479, 2018.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

F. Scarselli, M. Gori, A. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

Murat Cihan Sorkun, Abhishek Khetan, and Süleyman Er. Aqsoldb, a curated reference set of aqueous solubility and 2d descriptors for a diverse set of compounds. *Scientific data*, 6 (1):1–8, 2019.

Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between node embeddings and structural graph representations. *International Conference on Learning Representations*, 2020.

Sainbayar Sukhbaatar, arthur szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2244–2252. 2016.

Shyam A Tailor, Javier Fernandez-Marques, and Nicholas D Lane. Degree-quant: Quantization-aware training for graph neural networks. *arXiv preprint arXiv:2008.05000*, 2020.

Shyam A Tailor, Felix Opolka, Pietro Lio, and Nicholas Donald Lane. Do we need anisotropic graph neural networks? In *International Conference on Learning Representations*, 2021.

Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Ran-gnns: breaking the capacity limits of graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

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

Clement Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing, 2020.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4): 395–416, 2007.

Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations*, 2022.

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.

Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Lukas M Weber, Wouter Saelens, Robrecht Cannoodt, Charlotte Soneson, Alexander Hapfelmeier, Paul P Gardner, Anne-Laure Boulesteix, Yvan Saeys, and Mark D Robinson. Essential guidelines for computational method benchmarking. *Genome biology*, 20(1):125, 2019.

Qiang Wei and Guangmin Hu. Evaluating graph neural networks under graph sampling scenarios. *PeerJ Computer Science*, 8:e901, 2022.

Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

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

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34, 2021.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *International Conference on Machine Learning*, 2019.

Haimin Zhang, Min Xu, Guoqiang Zhang, and Kenta Niwa. Ssfg: Stochastically scaling features and gradients for regularizing graph convolutional networks. *arXiv preprint arXiv:2102.10338*, 2021.

Wentao Zhao, Dalin Zhou, Xinguo Qiu, and Wei Jiang. A pipeline for fair comparison of graph neural networks in node classification tasks, 2020.

Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *Advances in Neural Information Processing Systems*, 33:4917–4928, 2020.

Appendix A. Related Work

In the last few years, graph neural networks (GNNs) have seen a great surge of interest with promising methods being developed for myriad of domains including chemistry (Duvenaud et al., 2015; Gilmer et al., 2017), physics (Cranmer et al., 2019; Sanchez-Gonzalez et al., 2020), social sciences (Kipf and Welling, 2017; Monti et al., 2019), knowledge graphs (Schlichtkrull et al., 2018; Chami et al., 2020), recommendation (Monti et al., 2017b; Ying et al., 2018), and neuroscience (Griffa et al., 2017). Historically, three classes of GNNs have been developed. The first models (Scarselli et al., 2009; Bruna et al., 2013; Defferrard et al., 2016; Sukhbaatar et al., 2016; Kipf and Welling, 2017; Hamilton et al., 2017) aimed at extending the original convolutional neural networks (LeCun et al., 1995, 1998) to graphs. The second class enhanced the original models with anisotropic operations on graphs (Perona and Malik, 1990), such as attention and gating mechanisms (Battaglia et al., 2016; Marcheggiani and Titov, 2017; Monti et al., 2017a; Veličković et al., 2018; Bresson and Laurent, 2017). The recent third class has introduced GNNs that improve upon theoretical limitations of previous models (Xu et al., 2019; Morris et al., 2019; Maron et al., 2019a; Chen et al., 2019b; Murphy et al., 2019; Srinivasan and Ribeiro, 2020). Specifically, the first two classes can only differentiate simple non-isomorphic graphs and cannot separate automorphic nodes. Developing powerful and theoretically expressive GNN architectures is a key concern towards practical applications and real-world adoption of graph machine learning. However, tracking recent progress has been challenging as most models are evaluated on small datasets such as Cora, Citeseer and TU, which are inappropriate to differentiate complex, simple and graph-agnostic architectures (Hoang and Maehara, 2019; Chen et al., 2019a), and do not have consensus on a unifying experimental setting (Errica et al., 2019; Hu et al., 2020).

Consequently, our motivation is to benchmark GNNs to identify and quantify what types of architectures, first principles or mechanisms are universal, generalizable, and scalable when we move to larger and more challenging datasets. Benchmarking provides a strong paradigm to answer these fundamental questions. It has proved to be beneficial for driving progress, identifying essential ideas, and solving domain-specific problems in several areas of science (Weber et al., 2019). Recently, the famous 2012 ImageNet challenge (Deng et al., 2009) has provided a benchmark dataset that has triggered the deep learning revolution (Krizhevsky et al., 2012; Malik, 2017). Nevertheless, designing successful benchmarks is highly challenging as it requires both a coding framework with a rigorous experimental setting for fair comparisons, all while being reproducible, as well as using appropriate datasets that can statistically separate model performance. The lack of benchmarks has been a major issue in GNN literature as the aforementioned requirements have not been rigorously enforced.

Appendix B. Graph Neural Network Pipeline

In this section, we describe the experimental pipeline for the two broad classes of GNN architectures that are benchmarked in this framework as representative GNN classes – Message Passing Graph Convolutional Networks (MP-GCNs), which are based on the message passing framework formalized in Gilmer et al. (2017), and Weisfeiler Lehman GNNs (WL-GNNs), which improves the theoretical limitations of MP-GCNs and align expressivity power to the

WL-tests to distinguish non-isomorphic graphs. The two pipelines are illustrated in Figure 3 for **GCNs** and Figure 4 for **WL-GNNs**.

In Section B.1, we describe the components of the setup of the GCN class with *vanilla* GCN (Kipf and Welling, 2017), GraphSage (Hamilton et al., 2017), MoNet (Monti et al., 2017a), GAT (Veličković et al., 2018), and GatedGCN (Bresson and Laurent, 2017), including the input layers, the GNN layers and the task based MLP classifier layers. We also include the description of GIN (Xu et al., 2019) in this section as this model can be interpreted as a GCN, although it was designed to differentiate non-isomorphic graphs. In Section B.2, we present the GNN layers and the task based MLP classifier layers for the class of WL-GNN models with Ring-GNNs (Chen et al., 2019b) and 3WL-GNNs (Maron et al., 2019a).

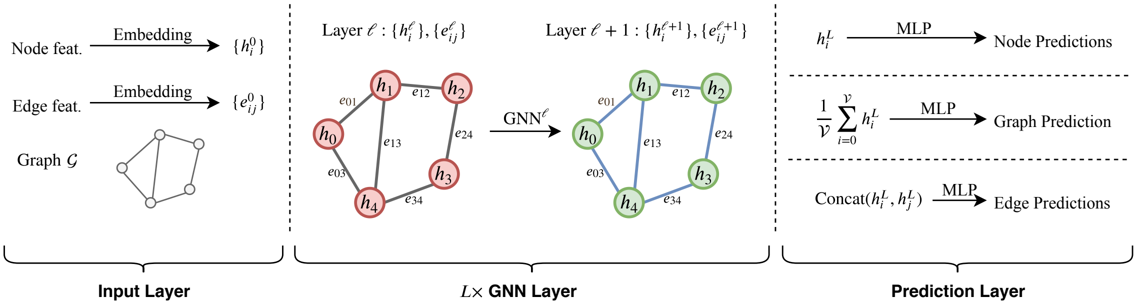


Figure 3: A standard experimental pipeline for GCNs, which embeds the graph node and edge features, performs several GNN layers to compute convolutional features, and finally makes a prediction through a task-specific MLP layer.

B.1 Message-Passing GCNs

For this class, we consider the widely used message passing-based graph convolutional networks (MP-GCNs), which update node representations from one layer to the other according to the formula: $h_i^{\ell+1} = f(h_i^\ell, \{h_j^\ell\}_{j\in\mathcal{N}_i})$. Note that the update equation is *local*, only depending on the neighborhood $\mathcal{N}_i$ of node i, and *independent of graph size*, making the space/time complexity $O(E)$ reducing to $O(n)$ for sparse graphs. Thus, MP-GCNs are highly parallelizable on GPUs and are implemented via sparse matrix multiplications in modern graph machine learning frameworks (Wang et al., 2019; Fey and Lenssen, 2019). MP-GCNs draw parallels to ConvNets for computer vision (LeCun et al., 1998) by considering a convolution operation with shared weights across the graph domain.

B.1.1 Input Layer

Given a graph, we are given node features $\alpha_i \in \mathbb{R}^{a\times 1}$ for each node i and (optionally) edge features $\beta_{ij} \in \mathbb{R}^{b\times 1}$ for each edge connecting node i and node j. The input features α_i and β_{ij} are embedded to d-dimensional hidden features $h_i^{\ell=0}$ and $e_{ij}^{\ell=0}$ via a simple linear projection before passing them to a graph neural network:

$$h_i^0 = U^0\alpha_i + u^0 \;\; ; \;\; e_{ij}^0 = V^0\beta_{ij} + v^0, \tag{1}$$

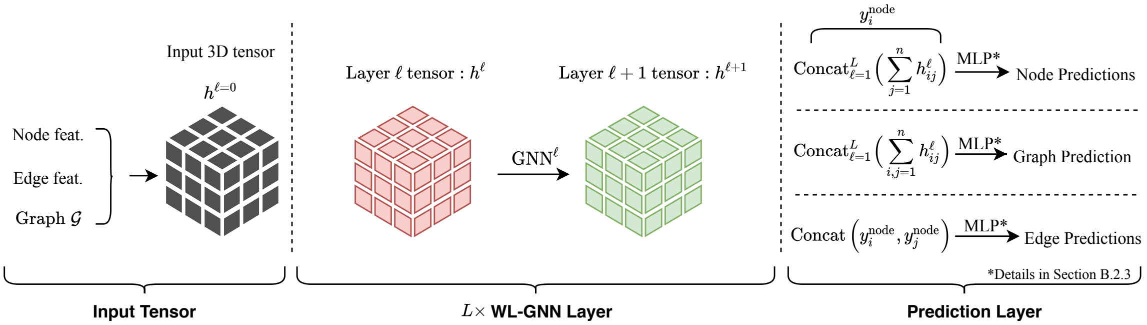

Input 3D tensor
$h^{\ell=0}$
Node feat.
Edge feat.
Graph $\mathcal{G}$
Layer ℓ tensor : h^{ℓ}
GNN$^{\ell}$
Layer $\ell+1$ tensor : $h^{\ell+1}$
y_i^{node}
$\text{Concat}_{\ell=1}^{L}\left(\sum_{j=1}^{n} h_{ij}^{\ell}\right) \xrightarrow{\text{MLP*}}$ Node Predictions
$\text{Concat}_{\ell=1}^{L}\left(\sum_{i,j=1}^{n} h_{ij}^{\ell}\right) \xrightarrow{\text{MLP*}}$ Graph Prediction
$\text{Concat}\left(y_i^{\text{node}}, y_j^{\text{node}}\right) \xrightarrow{\text{MLP*}}$ Edge Predictions
*Details in Section B.2.3
Input Tensor
$L\times$ WL-GNN Layer
Prediction Layer

Figure 4: A standard experimental pipeline for WL-GNNs, which inputs to a GNN a graph with all node and edge information (if available) represented by a dense tensor, performs several GNN layer computations over the dense tensor, and finally makes a prediction through a task-specific MLP layer.

where $U^0 \in \mathbb{R}^{d\times a}$, $V^0 \in \mathbb{R}^{d\times b}$ and $u^0, v^0 \in \mathbb{R}^d$. If the input node/edge features are one-hot vectors of discrete variables, then biases u^0, v^0 are not used.

B.1.2 GCN layers

Each GCN layer computes d-dimensional representations for the nodes/edges of the graph through recursive neighborhood diffusion (or message passing), where each graph node gathers features from its neighbors to represent local graph structure. Stacking L GCN layers allows the network to build node representations from the L-hop neighborhood of each node.

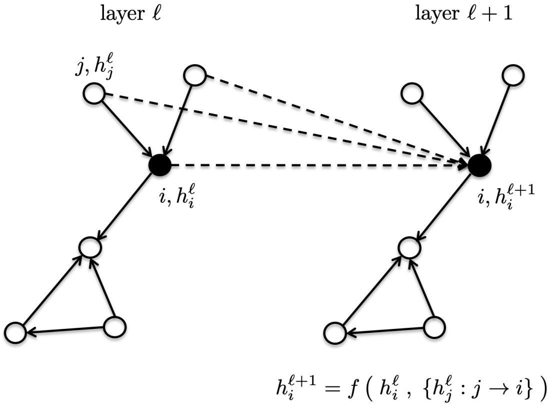


Figure 5: A generic graph neural network layer. Figure adapted from Bresson and Laurent (2017).

Let h_i^{ℓ} denote the feature vector at layer ℓ associated with node i. The updated features $h_i^{\ell+1}$ at the next layer $\ell+1$ are obtained by applying non-linear transformations to the central feature vector h_i^{ℓ} and the feature vectors h_j^{ℓ} for all nodes j in the neighborhood of node i (defined by the graph structure). This guarantees the transformation to build local reception fields, such as in standard ConvNets for computer vision, and be invariant to both graph size and vertex re-indexing.

Thus, the most generic version of a feature vector $h_i^{\ell+1}$ at vertex i at the next layer in the GNN is:

$$h_i^{\ell+1} = f\Big(\; h_i^{\ell} \;,\; \{h_j^{\ell} : j \to i\} \;\Big), \tag{2}$$

where $\{j \to i\}$ denotes the set of neighboring nodes j pointed to node i, which can be replaced by $\{j \in \mathcal{N}_i\}$, the set of neighbors of node i, if the graph is undirected. In other words, a GNN is defined by a mapping f taking as input a vector h_i^{ℓ} (the feature vector of the center vertex) as well as an un-ordered set of vectors $\{h_j^{\ell}\}$ (the feature vectors of all neighboring vertices), see Figure 5. The arbitrary choice of the mapping f defines an instantiation of a class of GNNs.

Graph ConvNets (GCN) (Kipf and Welling, 2017) In the simplest formulation of GNNs, *vanilla* Graph ConvNets iteratively update node features via an isotropic averaging operation over the neighborhood node features, *i.e.*,

$$\begin{aligned} h_i^{\ell+1} &= \text{ReLU}\Big(U^{\ell} \; \text{Mean}_{j\in\mathcal{N}_i} \; h_j^{\ell}\Big), &(3)\\ &= \text{ReLU}\Big(U^{\ell} \; \frac{1}{\deg_i} \sum_{j\in\mathcal{N}_i} \; h_j^{\ell}\Big), &(4)\end{aligned}$$

where $U^{\ell} \in \mathbb{R}^{d\times d}$ (a bias is also used, but omitted for clarity purpose), $\deg_i$ is the in-degree of node i, see Figure 6. Eq. (3) is called a *convolution* as it is a linear approximation of a localized spectral convolution. Note that it is possible to add the central node features h_i^{ℓ} in the update (3) by using self-loops or residual connections.

The GCN model in Kipf and Welling (2017) use symmetric normalization instead of the isotropic averaging, to result in the following node update equation:

$$h_i^{\ell+1} = \text{ReLU}\Big(U^{\ell} \; \frac{1}{\sqrt{\deg_i}\sqrt{\deg_j}} \sum_{j\in\mathcal{N}_i} \; h_j^{\ell}\Big), \tag{5}$$

GraphSage (Hamilton et al., 2017) GraphSage improves upon the simple GCN model by explicitly incorporating each node's own features from the previous layer in its update equation:

$$\hat{h}_i^{\ell+1} = \text{ReLU}\Big(U^{\ell} \; \text{Concat}\left(h_i^{\ell}, \; \text{Mean}_{j\in\mathcal{N}_i} \; h_j^{\ell}\right)\Big), \tag{6}$$

where $U^{\ell} \in \mathbb{R}^{d\times 2d}$, see Figure 7. Observe that the transformation applied to the central node features h_i^{ℓ} is different to the transformation carried out to the neighborhood features h_j^{ℓ}. The node features are then projected onto the ℓ_2-unit ball before being passed to the next layer:

$$h_i^{\ell+1} = \frac{\hat{h}_i^{\ell+1}}{\|\hat{h}_i^{\ell+1}\|_2}. \tag{7}$$

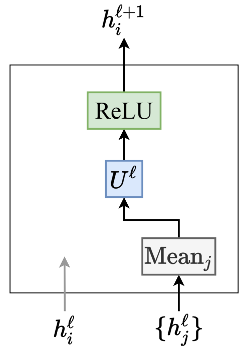


Figure 6: GCN Layer

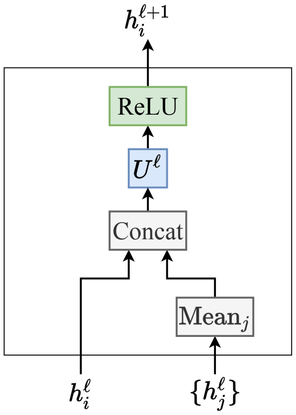


Figure 7: GraphSage Layer

The authors also define more sophisticated neighborhood aggregation functions, such as Max-pooling or LSTM aggregators:

$$\hat{h}_i^{\ell+1} = \text{ReLU}\Big(U^\ell \text{ Concat}\left(h_i^\ell,\ \text{Max}_{j\in\mathcal{N}_i}\text{ReLU}\left(V^\ell h_j^\ell\right)\right)\Big), \tag{8}$$

$$\hat{h}_i^{\ell+1} = \text{ReLU}\Big(U^\ell \text{ Concat}\left(h_i^\ell,\ \text{LSTM}^\ell_{j\in\mathcal{N}_i}\left(h_j^\ell\right)\right)\Big), \tag{9}$$

where $V^\ell \in \mathbb{R}^{d\times d}$ and the LSTM$^\ell$ cell also uses learnable weights. In our experiments, we use the Max-pooling version of GraphSage, Eq.(8).

Graph Attention Network (GAT) (Veličković et al., 2018) GAT uses an attention mechanism (Bahdanau et al., 2014) to introduce anisotropy in the neighborhood aggregation function. The network employs a multi-headed architecture to increase the learning capacity, similar to the Transformer (Vaswani et al., 2017; Joshi, 2020). The node update equation is given by:

$$h_i^{\ell+1} = \text{Concat}_{k=1}^K\Big(\text{ELU}\Big(\sum_{j\in\mathcal{N}_i} e_{ij}^{k,\ell}\ U^{k,\ell}\ h_j^\ell\Big)\Big), \tag{10}$$

where $U^{k,\ell} \in \mathbb{R}^{\frac{d}{K}\times d}$ are the K linear projection heads, and $e_{ij}^{k,\ell}$ are the attention coefficients for each head defined as:

$$e_{ij}^{k,\ell} = \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}_{ij'}^{k,\ell})}, \tag{11}$$

$$\hat{e}_{ij}^{k,\ell} = \text{LeakyReLU}\Big(V^{k,\ell}\ \text{Concat}\big(U^{k,\ell}h_i^\ell,\ U^{k,\ell}h_j^\ell\big)\Big), \tag{12}$$

where $V^{k,\ell} \in \mathbb{R}^{\frac{2d}{K}}$, see Figure 8. GAT learns a mean over each node's neighborhood features sparsely weighted by the importance of each neighbor.

MoNet (Monti et al., 2017a) The MoNet model introduces a general architecture to learn on graphs and manifolds using the Bayesian Gaussian Mixture Model (GMM) (Dempster

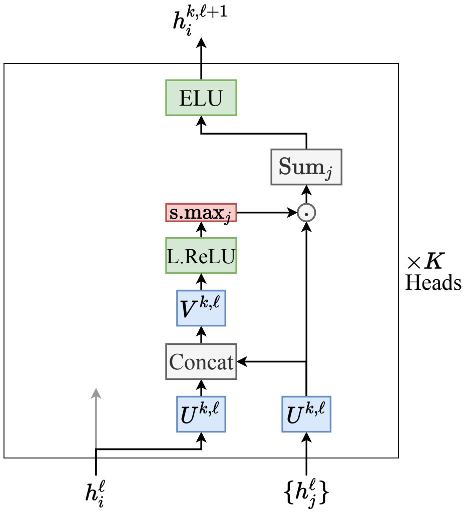


Figure 8: GAT Layer

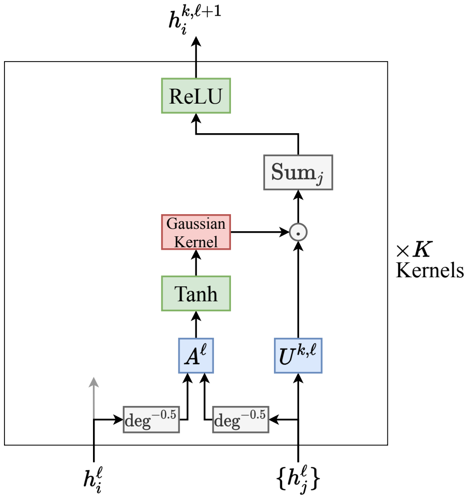


Figure 9: MoNet Layer

et al., 1977). In the case of graphs, the node update equation is defined as:

$$h_i^{\ell+1} \;=\; \text{ReLU}\Big(\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} e_{ij}^{k,\ell}\; U^{k,\ell}\; h_j^\ell\Big), \tag{13}$$

$$e_{ij}^{k,\ell} \;= \exp\Big(-\frac{1}{2}(u_{ij}^\ell-\mu_k^\ell)^T(\Sigma_k^\ell)^{-1}(u_{ij}^\ell-\mu_k^\ell)\Big), \tag{14}$$

$$u_{ij}^\ell \;=\; \text{Tanh}\Big(A^\ell(\text{deg}_i^{-1/2},\text{deg}_j^{-1/2})^T + a^\ell\Big), \tag{15}$$

where $U^{k,\ell}\in\mathbb{R}^{d\times d}$, $\mu_k^\ell, (\Sigma_k^\ell)^{-1}, a^\ell\in\mathbb{R}^2$ and $A^\ell\in\mathbb{R}^{2\times 2}$ are the (learnable) parameters of the GMM, see Figure 9.

Gated Graph ConvNet (GatedGCN) (Bresson and Laurent, 2017) GatedGCN considers residual connections, batch normalization and edge gates (Marcheggiani and Titov, 2017) to design another anisotropic variant of GCN. The authors propose to explicitly update edge features along with node features:

$$h_i^{\ell+1} = h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell h_i^\ell + \sum_{j\in\mathcal{N}_i} e_{ij}^\ell \odot V^\ell h_j^\ell\Big)\Big), \tag{16}$$

where $U^\ell, V^\ell\in\mathbb{R}^{d\times d}$, $\odot$ is the Hadamard product, and the edge gates e_{ij}^ℓ are defined as:

$$e_{ij}^\ell \;=\; \frac{\sigma(\hat{e}_{ij}^\ell)}{\sum_{j'\in\mathcal{N}_i}\sigma(\hat{e}_{ij'}^\ell)+\varepsilon}, \tag{17}$$

$$\hat{e}_{ij}^\ell \;=\; \hat{e}_{ij}^{\ell-1} + \text{ReLU}\Big(\text{BN}\big(A^\ell h_i^{\ell-1} + B^\ell h_j^{\ell-1} + C^\ell \hat{e}_{ij}^{\ell-1}\big)\Big), \tag{18}$$

where σ is the sigmoid function, ε is a small fixed constant for numerical stability, $A^\ell, B^\ell, C^\ell\in\mathbb{R}^{d\times d}$, see Figure 10. Note that the edge gates (17) can be regarded as a soft attention process,

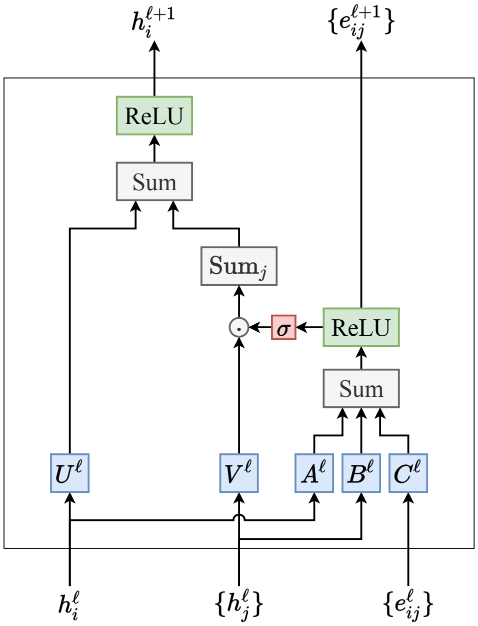


Figure 10: GatedGCN Layer

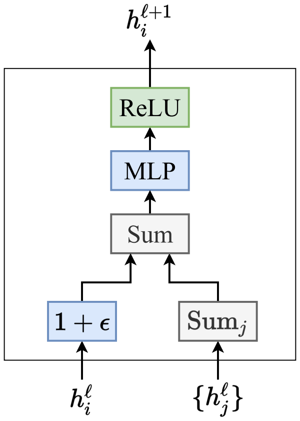


Figure 11: GIN Layer

related to the standard sparse attention mechanism (Bahdanau et al., 2014). Different from other anisotropic GNNs, the GatedGCN architecture explicitly maintains edge features $\hat{e}_{ij}$ at each layer, following Bresson and Laurent (2019); Joshi et al. (2019).

Graph Isomorphism Networks (GIN) (Xu et al., 2019) The GIN architecture is based the Weisfeiler-Lehman Isomorphism Test (Weisfeiler and Lehman, 1968) to study the expressive power of GNNs. The node update equation is defined as:

$$h_i^{\ell+1} = \text{ReLU}\Big(U^\ell \big(\text{ReLU}\big(\text{BN}\big(V^\ell \hat{h}_i^{\ell+1}\big) \big) \big) \Big), \tag{19}$$

$$\hat{h}_i^{\ell+1} = (1+\epsilon)\, h_i^\ell + \sum_{j \in \mathcal{N}_i} h_j^\ell, \tag{20}$$

where ϵ is a learnable constant, $U^\ell, V^\ell \in \mathbb{R}^{d \times d}$, BN denotes Batch Normalization. See Figure 11 for illustration of the update equation.

Normalization and Residual Connections As a final note, we augment each message-passing GCN layer with batch normalization (BN) (Ioffe and Szegedy, 2015) and residual connections (He et al., 2016). As such, we consider a more specific class of GCNs than (2):

$$h_i^{\ell+1} = h_i^\ell + \sigma\Big(\text{BN}\left(\hat{h}_i^{\ell+1}\right)\Big), \tag{21}$$

$$\hat{h}_i^{\ell+1} = g_{\text{GCN}}\left(h_i^\ell \ , \ \{h_j^\ell : j \to i\} \right), \tag{22}$$

where σ is a non-linear activation function and g_{GCN} is a specific message-passing GCN layer.

B.1.3 Task-based Layer

The final component of each network is a prediction layer to compute task-dependent outputs, which are given to a loss function to train the network parameters in an end-to-end manner. The input of the prediction layer is the result of the final message-passing GCN layer for each node of the graph (except GIN, which uses features from all intermediate layers).

Graph classifier layer To perform graph classification, we first build a d-dimensional graph-level vector representation $y_{\mathcal{G}}$ by averaging over all node features in the final GCN layer:

$$y_{\mathcal{G}} = \frac{1}{\mathcal{V}} \sum_{i=0}^{\mathcal{V}} h_i^L, \tag{23}$$

The graph features are then passed to a MLP, which outputs un-normalized logits/scores $y_{\text{pred}} \in \mathbb{R}^C$ for each class:

$$y_{\text{pred}} = P \ \text{ReLU} \left(Q \ y_{\mathcal{G}} \right), \tag{24}$$

where $P \in \mathbb{R}^{d \times C}, Q \in \mathbb{R}^{d \times d}, C$ is the number of classes. Finally, we minimize the cross-entropy loss between the logits and groundtruth labels.

Graph regression layer For graph regression, we compute $y_{\mathcal{G}}$ using Eq.(23) and pass it to a MLP which gives the prediction score $y_{\text{pred}} \in \mathbb{R}$:

$$y_{\text{pred}} = P \ \text{ReLU} \left(Q \ y_{\mathcal{G}} \right), \tag{25}$$

where $P \in \mathbb{R}^{d \times 1}, Q \in \mathbb{R}^{d \times d}$. The L1-loss between the predicted score and the groundtruth score is minimized during the training.

Node classifier layer For node classification, we independently pass each node's feature vector to a MLP for computing the un-normalized logits $y_{i,\text{pred}} \in \mathbb{R}^C$ for each class:

$$y_{i,\text{pred}} = P \ \text{ReLU} \left(Q \ h_i^L \right), \tag{26}$$

where $P \in \mathbb{R}^{d \times C}, Q \in \mathbb{R}^{d \times d}$. The cross-entropy loss weighted inversely by the class size is used during training.

Edge classifier layer To make a prediction for each graph edge e_{ij}, we first concatenate node features h_i and h_j from the final GNN layer. The concatenated edge features are then passed to a MLP for computing the un-normalized logits $y_{ij,\text{pred}} \in \mathbb{R}^C$ for each class:

$$y_{ij,\text{pred}} = P \ \text{ReLU} \left(Q \ \text{Concat} \left(h_i^L, \ h_j^L \right) \right), \tag{27}$$

where $P \in \mathbb{R}^{d \times C}, Q \in \mathbb{R}^{d \times 2d}$. The standard cross-entropy loss between the logits and groundtruth labels is used.

B.2 Weisfeiler-Lehman GNNs

Weisfeiler-Lehman GNNs are the second GNN class we include in our benchmarking framework which are based on the WL test (Weisfeiler and Lehman, 1968). Xu et al. (2019) introduced GIN–Graph Isomorphism Network, a provable 1-WL GNN, which can distinguish two non-isomorphic graphs w.r.t. the 1-WL test. Higher k-WL isomorphic tests lead to more

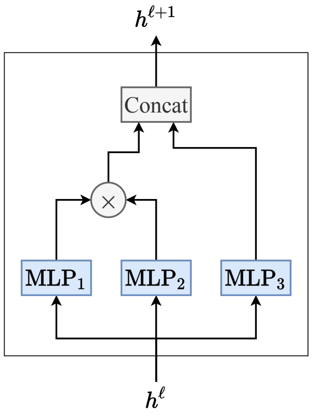


Figure 12: 3WL-GNN Layer

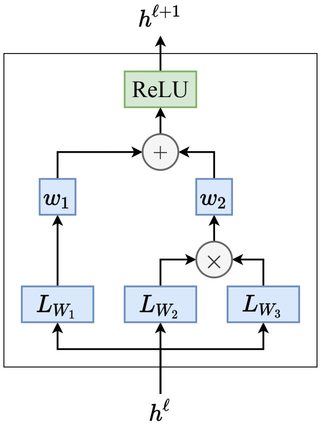


Figure 13: RingGNN Layer

discriminative k-WL GNNs in (Morris et al., 2019; Maron et al., 2019a). However, k-WL GNNs require the use of tensors of rank k, which is intractable in practice for $k > 2$. As a result, Maron et al. (2019a) proposed a model, namely 3-WL GNNs, that uses rank-2 tensors while being 3-WL provable. This 3-WL model improves the space/time complexities of Morris et al. (2019) from $O(n^3)/O(n^4)$ to $O(n^2)/O(n^3)$ respectively. We use 3WLGNNs (Maron et al., 2019a) and RingGNNs (Chen et al., 2019b) as the GNN instances in this class, the experimental pipeline of which are described as follows.

B.2.1 Input Tensor

For a given graph with adjacency matrix $A \in \mathbb{R}^{n\times n}$, node features $h^{\text{node}} \in \mathbb{R}^{n\times d}$ and edge features $h^{\text{edge}} \in \mathbb{R}^{n\times n\times d_e}$, the input tensor to the RingGNN and 3WL-GNN networks is defined as

$$h^{\ell=0} \in \mathbb{R}^{n\times n\times(1+d+d_e)}, \tag{28}$$

where

$$h^{\ell=0}_{i,j,1} = A_{ij} \in \mathbb{R}, \quad \forall i,j \tag{29}$$

$$h^{\ell=0}_{i,j,2:d+1} = \begin{cases} h^{\text{node}}_i \in \mathbb{R}^d, & \forall i = j \\ 0 \qquad\quad , & \forall i \neq j \end{cases} \tag{30}$$

$$h^{\ell=0}_{i,j,d+2:d+d_e+1} = h^{\text{edge}}_{ij} \in \mathbb{R}^{d_e} \tag{31}$$

B.2.2 WL-GNN layers

3WL-GNNs (Maron et al., 2019a) These networks introduced an architecture that can distinguish two non-isomorphic graphs with the 3-WL test. The layer update equation of 3WL-GNNs is defined as:

$$h^{\ell+1} = \text{Concat}\Big(M_{W_1^\ell}(h^\ell) \ . \ M_{W_2^\ell}(h^\ell), \ M_{W_3^\ell}(h^\ell)\Big), \tag{32}$$

where $h^\ell, h^{\ell+1} \in \mathbb{R}^{n\times n\times d}$, and M_W are 2-layer MLPs applied along the feature dimension:

$$M_{W=\{W_a,W_b\}}(h) = \sigma\Big(h\ W_a\Big)W_b, \tag{33}$$

where $W_a, W_b \in \mathbb{R}^{d\times d}$. As $h \in \mathbb{R}^{n\times n\times d}$, the MLP (33) is implemented with a standard 2D-convolutional layer with 1×1 kernel size. Eventually, the matrix multiplication in (32) is carried out along the first and second dimensions such that:

$$\Big(M_{W_1}(h)\ .\ M_{W_2}(h)\Big)_{i,j,k} = \sum_{p=1}^{n} \Big(M_{W_1}(h)\Big)_{i,p,k}\ .\ \Big(M_{W_2}(h)\Big)_{p,j,k}, \tag{34}$$

with complexity $O(n^3)$.

Ring-GNNs (Chen et al., 2019b) These models proposed to improve the order-2 equivariant GNNs of Maron et al. (2019b) with the multiplication of two equivariant linear layers. The layer update equation of Ring-GNNs is designed as:

$$h^{\ell+1} = \sigma\Big(w_1^\ell\ L_{W_1^\ell}(h^\ell) + w_2^\ell\ L_{W_2^\ell}(h^\ell).L_{W_3^\ell}(h^\ell)\Big), \tag{35}$$

where $h^\ell, h^{\ell+1} \in \mathbb{R}^{n\times n\times d}$, $w_{1,2}^\ell \in \mathbb{R}$, and L_W are the equivariant linear layers defined as

$$\Big(L_W(h)\Big)_{i,j,k} = \sum_{p=1}^{17}\sum_{q=1}^{d} W_{p,q,k}\Big(L_i(h)\Big)_{i,j,q}, \tag{36}$$

where $W \in \mathbb{R}^{d\times d\times 17}$ and $\{L_i\}_{i=1}^{15}$ is the set of all basis functions for all linear equivariant functions from $\mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ (see Appendix A in Maron et al. (2019b) for the complete list of these 15 operations) and $\{L_i\}_{i=16}^{17}$ are the basis for the bias terms. Matrix multiplication in (35) also implies a time complexity $O(n^3)$.

B.2.3 Task-based network layers

We describe the final network layers depending on the task at hand. The loss functions corresponding to the task are the same as the GCNs, and presented in Section B.1.3.

Graph classifier layer We have followed the original author implementations in Maron et al. (2019a,b); Chen et al. (2019b) to design the classifier layer for 3WL-GNNs and Ring-GNNs. Similar to Xu et al. (2018, 2019), the classifier layer for Ring-GNNs uses features from all intermediate layers and then passes the features to a MLP:

$$y_{\mathcal{G}} \ = \ \text{Concat}_{\ell=1}^{L}\Big(\sum_{i,j=1}^{n} h_{ij}^\ell\Big)\ \in \mathbb{R}^{Ld}, \tag{37}$$

$$y_{\text{pred}} \ = \ P\ \text{ReLU}\,(Q\ y_{\mathcal{G}})\ \in \mathbb{R}^{C}, \tag{38}$$

where $P \in \mathbb{R}^{d\times C}, Q \in \mathbb{R}^{Ld\times d}, C$ is the number of classes.

For 3WL-GNNs, Eqn. (37) is replaced by a diagonal and off-diagonal max pooling readout Maron et al. (2019a,b) at every layer:

$$y_{\mathcal{G}}^\ell \ = \ \text{Concat}\Big(\max_i\ h_{ii}^\ell\ ,\ \max_{i\neq j}\ h_{ij}^\ell\Big)\ \in \mathbb{R}^{2d}, \tag{39}$$

and the final prediction score is defined as:

$$y_{\text{pred}} \;=\; \sum_{\ell=1}^{L} P^{\ell} y_{\mathcal{G}}^{\ell} \;\in \mathbb{R}^{C}, \tag{40}$$

where $P^{\ell} \in \mathbb{R}^{2d \times C}$, C is the number of classes.

Graph regression layer Similar to the graph classifier layer with $P \in \mathbb{R}^{d \times 1}$ for Ring-GNNs, and $P^{\ell} \;\in \mathbb{R}^{2d \times 1}$ for 3WL-GNNs.

Node classifier layer For node classification, the prediction in Ring-GNNs is done as follows:

$$y_i^{\text{node}} \;=\; \text{Concat}_{\ell=1}^{L}\Big(\sum_{j=1}^{n} h_{ij}^{\ell}\Big) \;\in \mathbb{R}^{Ld}, \tag{41}$$

$$y_{i,\text{pred}} \;=\; P\ \text{ReLU}\left(Q\ y_i^{\text{node}}\right) \;\in \mathbb{R}^{C}, \tag{42}$$

where $P \in \mathbb{R}^{d \times C}, Q \in \mathbb{R}^{Ld \times d}, C$ is the number of classes.

In 3WL-GNNs, the final prediction score is defined as:

$$y_i^{\text{node},\ell} \;=\; \sum_{j=1}^{n} h_{ij}^{\ell} \;\in \mathbb{R}^{d}, \tag{43}$$

$$y_{i,\text{pred}} \;=\; \sum_{\ell=1}^{L} P^{\ell}\ y_i^{\text{node},\ell} \;\in \mathbb{R}^{C}, \tag{44}$$

where $P^{\ell} \in \mathbb{R}^{d \times C}, C$ is the number of classes.

Edge classifier layer For link prediction, for both Ring-GNNs and 3WL-GNNs, the edge features are obtained by concatenating the node features such as:

$$y_i^{\text{node}} \;=\; \text{Concat}_{\ell=1}^{L}\Big(\sum_{j=1}^{n} h_{ij}^{\ell}\Big) \;\in \mathbb{R}^{Ld}, \tag{45}$$

$$y_{ij,\text{pred}} \;=\; P\ \text{ReLU}\left(Q\ \text{Concat}\left(y_i^{\text{node}}, y_j^{\text{node}}\right)\right) \;\in \mathbb{R}^{C}, \tag{46}$$

where $P \in \mathbb{R}^{d \times C}, Q \in \mathbb{R}^{2Ld \times d}, C$ is the number of classes.

Appendix C. Datasets and Benchmarking Experiments

In this section, we provide details on the datasets included in the benchmarking framework (Table 1) and the numerical results of the experiments using the GNN described in Section B, which also consists experiments from a simple graph-agnostic baseline for every dataset that parallelly applies an MLP on each node's feature vector, independent of other nodes. For complete statistics of the data, see Table 2. The experimental overview in terms of the training strategy, reporting of results and the parameter budget used for fair comparison are described first, as follows.

Dataset	**#Graphs**	**#Classes**	**Avg. Nodes**	**Avg. Edges**	**Node feat. (dim)**	**Edge feat. (dim)**
ZINC	12000	–	23.16	49.83	Atom Type (28)	Bond Type (4)
AQSOL	9823	–	17.57	35.76	Atom Type (65)	Bond Type (5)
OGBL-COLLAB	1	–	235868.00	2358104.00	Word Embs (128)	Year & Weight (2)
WikiCS	1	10	11701.00	216123.00	Word Embs (300)	N.A.
MNIST	70000	10	70.57	564.53	Pixel+Coord (3)	Node Dist (1)
CIFAR10	60000	10	117.63	941.07	Pixel[RGB]+Coord (5)	Node Dist (1)
PATTERN	14000	2	117.47	4749.15	Node Attr (3)	N.A.
CLUSTER	12000	6	117.20	4301.72	Node Attr (7)	N.A.
TSP	12000	2	275.76	6894.04	Coord (2)	Node Dist (1)
CSL	150	10	41.00	164.00	N.A.	N.A.
CYCLES	20000	2	48.96	87.84	N.A.	N.A.
GraphTheoryProp	7040	–	18.82	95.17	Source S.P.+Random(2)	N.A.

Table 2: Summary statistics of all datasets. Numbers in parentheses of Node features and Edge features are the dimensions. S.P. denotes shortest path.

Training. We use the Adam optimizer (Kingma and Ba, 2014) with the same learning rate decay strategy for all models. An initial learning rate is selected in $\{10^{-2}, 10^{-3}, 10^{-4}\}$ which is reduced by half if the validation loss does not improve after a fixed number of epochs, in the range 5-25. We do not set a maximum number of epochs – the training is stopped either when the learning rate has reached the small value of 10^{-6}, or the computational time reaches 12 hours. We run each experiment with 4 different seeds and report the statistics of the 4 results. More details are provided in each experimental sub-sections.

Task-based network layer. The node representations generated by the final layer of GCNs, or the dense tensor obtained at the final layer of the higher order WL-GNNs, are passed to a network suffix which is usually a downstream MLP of 3 layers. For GIN, RingGNN, and 3WL-GNN, we follow the original instructions of network suffixes to consider feature outputs from each layer of the network, similar to that of Jumping Knowledge Networks (Xu et al., 2018). Refer to the equations in the Sections B.1.3 and B.2.3 for more details.

Parameter budgets. Our goal is not to find the optimal set of hyperparameters for a specific GNN model (which is computationally expensive), but to compare and benchmark the model and/or their building blocks within a budget of parameters. Therefore, we decide on using two parameter budgets: (1) 100k parameters for each GNNs for all the tasks, and (2) 500k parameters for GNNs for which we investigate scaling a model to larger parameters and deeper layers. The number of hidden layers and hidden dimensions are selected accordingly to match these budgets. The configuration details of each single experiment can be found in our modular coding infrastructure on GitHub.

C.1 Graph Regression with ZINC dataset

For the ZINC dataset in our benchmark, we use a subset (12K) of ZINC molecular graphs (250K) dataset (Irwin et al., 2012) to regress a molecular property known as the constrained solubility which is the term log$P$ − SA − cycle (octanol-water partition coefficients, log$P$, penalized by the synthetic accessibility score, SA, and number of long cycles, cycle). For each molecular graph, the node features are the types of heavy atoms and the edge features

		ZINC				
Model	L	**#Param**	**Test MAE±s.d.**	**Train MAE±s.d.**	**#Epoch**	**Epoch/Total**
MLP	4	108975	0.706±0.006	0.644±0.005	116.75	1.01s/0.03hr
vanilla GCN	4	103077	0.459±0.006	0.343±0.011	196.25	2.89s/0.16hr
	16	505079	0.367±0.011	0.128±0.019	197.00	12.78s/0.71hr
GraphSage	4	94977	0.468±0.003	0.251±0.004	147.25	3.74s/0.15hr
	16	505341	0.398±0.002	0.081±0.009	145.50	16.61s/0.68hr
GCN	4	103077	0.416±0.006	0.313±0.011	159.50	1.53s/0.07hr
	16	505079	0.278±0.003	0.101±0.011	159.25	3.66s/0.16hr
MoNet	4	106002	0.397±0.010	0.318±0.016	188.25	1.97s/0.10hr
	16	504013	0.292±0.006	0.093±0.014	171.75	10.82s/0.52hr
GAT	4	102385	0.475±0.007	0.317±0.006	137.50	2.93s/0.11hr
	16	531345	0.384±0.007	0.067±0.004	144.00	12.98s/0.53hr
GatedGCN	4	105735	0.435±0.011	0.287±0.014	173.50	5.76s/0.28hr
GatedGCN-E	4	105875	0.375±0.003	0.236±0.007	194.75	5.37s/0.29hr
	16	504309	0.282±0.015	0.074±0.016	166.75	20.50s/0.96hr
GatedGCN-E-PE	16	505011	**0.214±0.013**	0.067±0.019	185.00	10.70s/0.56hr
GIN	4	103079	0.387±0.015	0.319±0.015	153.25	2.29s/0.10hr
	16	509549	0.526±0.051	0.444±0.039	147.00	10.22s/0.42hr
RingGNN	2	97978	0.512±0.023	0.383±0.020	90.25	327.65s/8.32hr
RingGNN-E	2	104403	0.363±0.026	0.243±0.025	95.00	366.29s/9.76hr
	2	527283	0.353±0.019	0.236±0.019	79.75	293.94s/6.63hr
	8	510305	Diverged	Diverged	Diverged	Diverged
3WLGNN	3	102150	0.407±0.028	0.272±0.037	111.25	286.23s/8.88hr
3WLGNN-E	3	103098	0.256±0.054	0.140±0.044	117.25	334.69s/10.90hr
	3	507603	0.303±0.068	0.173±0.041	120.25	329.49s/11.08hr
	8	582824	0.303±0.057	0.246±0.043	52.50	811.27s/12.15hr

Table 3: Benchmarking results for ZINC for graph regression. Results (lower is better) are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models. The suffix -E denotes the use of available edge features, and the suffix -PE denote the use of Laplacian Eigenvectors as node positional encodings with dimension 8.

are the types of bonds between them. ZINC has been used popularly for research related to molecular graph generation (Jin et al., 2018; Bresson and Laurent, 2019).

Splitting. ZINC has $10,000$ train, $1,000$ validation and $1,000$ test graphs.

Training. For the learning rate strategy across all GNNs, an initial learning rate is set to 1×10^{-3}, the reduce factor is 0.5, and the stopping learning rate is 1×10^{-5}. The patience value is 5 for 3WLGNN and RingGNN, and 10 for all other GNNs.

Performance Measure. The performance measure is the mean absolute error (MAE) between the predicted and the groundtruth constrained solubility for each molecular graph.

Results. The numerical results are presented in Table 3 and analysed in Section D collectively with other benchmarking results.

C.2 Graph Regression with AQSOL dataset

AQSOL dataset is based on AqSolDB (Sorkun et al., 2019) which is a standardized database of $9,982$ molecular graphs with their aqueous solubility values, collected from 9 different data sources. The aqueous solubility targets are collected from experimental measurements and standardized to Log$S$ units in AqSolDB. We use these final values as the property to regress in the AQSOL dataset which is the resultant collection after we filter out few graphs with no edges (bonds) and a small number of graphs with missing node feature values. Thus, the total molecular graphs are $9,823$. For each molecular graph, the node features are the types of heavy atoms and the edge features are the types of bonds between them.

Splitting. We provide a scaffold splitting (Hu et al., 2020) of the dataset in the ratio $8:1:1$ to have $7,831$ train, 996 validation and 996 test graphs.

Training. For the learning rate strategy across all GNNs, an initial learning rate is set to 1×10^{-3}, the reduce factor is 0.5, and the stopping learning rate is 1×10^{-5}. The patience value is 5 for 3WLGNN and RingGNN, and 10 for all other GNNs.

Performance Measure. Similar to ZINC, the performance measure is the mean absolute error (MAE) between the predicted and the actual aqueous solubility values.

Results. The numerical results are presented in Table 4 and analysed in Section D.

				AQSOL		
Model	L	**#Param**	**TestMAE±s.d.**	**TrainMAE±s.d.**	**Epochs**	**Epoch/Total**
MLP	4	114525	1.744±0.016	1.413±0.042	85.75	0.61s/0.02hr
vanilla GCN	4	108442	1.483±0.014	0.791±0.034	110.25	1.14s/0.04hr
	16	511443	1.458±0.011	0.567±0.027	121.50	2.83s/0.10hr
GraphSage	4	109620	1.431±0.010	0.666±0.027	106.00	1.51s/0.05hr
	16	509078	1.402±0.013	0.402±0.013	110.50	3.20s/0.10hr
GCN	4	108442	1.372±0.020	0.593±0.030	135.00	1.28s/0.05hr
	16	511443	1.333±0.013	0.382±0.018	137.25	3.31s/0.13hr
MoNet	4	109332	1.395±0.027	0.557±0.022	125.50	1.68s/0.06hr
	16	507750	1.501±0.056	0.444±0.024	110.00	3.62s/0.11hr
GAT	4	108289	1.441±0.023	0.678±0.021	104.50	1.92s/0.06hr
	16	540673	1.403±0.008	0.386±0.014	111.75	4.44s/0.14hr
GatedGCN	4	108325	1.352±0.034	0.576±0.056	142.75	2.28s/0.09hr
	16	507039	1.355±0.016	0.465±0.038	99.25	5.52s/0.16hr
GatedGCN-E	4	108535	1.295±0.016	0.544±0.033	116.25	2.29s/0.08hr
	16	507273	1.308±0.013	0.367±0.012	110.25	5.61s/0.18hr
GatedGCN-E-PE	16	507663	**0.996±0.008**	0.372±0.016	105.25	5.70s/0.30hr
GIN	4	107149	1.894±0.024	0.660±0.027	115.75	1.55s/0.05hr
	16	514137	1.962±0.058	0.850±0.054	128.50	3.97s/0.14hr
RingGNN	2	116643	20.264±7.549	0.625±0.018	54.25	113.99s/1.76hr
RingGNN-E	2	123157	3.769±1.012	0.470±0.022	63.75	125.17s/2.26hr
	2	523935	Diverged	Diverged	Diverged	Diverged
	8	-	Diverged	Diverged	Diverged	Diverged
3WLGNN	3	110919	1.154±0.050	0.434±0.026	66.75	130.92s/2.48hr
	3	525423	1.108±0.036	0.405±0.031	70.75	131.12s/2.62hr
3WLGNN-E	3	112104	1.042±0.064	0.307±0.024	68.50	139.04s/2.70hr
	3	528123	1.052±0.034	0.287±0.023	67.00	140.43s/2.67hr
	8	-	Diverged	Diverged	Diverged	Diverged

Table 4: Benchmarking results for AQSOL for graph regression. Results (lower is better) are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models. The suffix -E denotes the use of available edge features, and the suffix -PE denote the use of Laplacian Eigenvectors as node positional encodings with dimension 4.

				OGBL-COLLAB		
Model	L	**#Param**	**Test Hits±s.d.**	**Train Hits±s.d.**	**#Epoch**	**Epoch/Total**
MLP	3	39441	20.350±2.168	29.807±3.360	147.50	2.09s/0.09hr
vanilla GCN	3	40479	50.422±1.131	92.112±0.991	122.50	351.05s/12.04hr
GraphSage	3	39856	51.618±0.690	99.949±0.052	152.75	277.93s/11.87hr
GCN	3	40479	48.956±1.143	87.385±2.056	142.25	7.66s/0.31hr
MoNet	3	39751	36.144±2.191	61.156±3.973	167.50	26.69s/1.26hr
GAT	3	42637	51.501±0.962	97.851±1.114	157.00	18.12s/0.80hr
GatedGCN	3	40965	52.635±1.168	96.103±1.876	95.00	453.47s/12.09hr
GatedGCN-PE	3	41889	**52.849±1.345**	96.165±0.453	94.75	452.75s/12.08hr
GatedGCN-E	3	40965	49.212±1.560	88.747±1.058	95.00	451.21s/12.03hr
GIN	3	39544	41.730±2.284	70.555±4.444	140.25	8.66s/0.34hr
RingGNN	-	-	OOM	*RingGNN and 3WLGNN rely on dense tensors which leads to* ***OOM*** *on both GPU and CPU memory.*		
3WLGNN	-	-	OOM			
Matrix Fact.	-	60546561	44.206±0.452	100.000±0.000	254.33	2.66s/0.21hr

Table 5: Benchmarking results for OGBL-COLLAB for link prediction. Results (higher is better) are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models. The suffix -E denotes the use of available edge features, and the suffix -PE denote the use of Laplacian Eigenvectors as node positional encodings with dimension 20.

C.3 Link Prediction with OGBL-COLLAB dataset

OGBL-COLLAB is a link prediction dataset proposed by OGB (Hu et al., 2020) corresponding to a collaboration network between approximately 235K scientists, indexed by Microsoft Academic Graph (Wang et al., 2020). Nodes represent scientists and edges denote collaborations between them. For node features, OGB provides 128-dimensional vectors, obtained by averaging the word embeddings of a scientist's papers. The year and number of co-authored papers in a given year are concatenated to form edge features. The graph can also be viewed as a dynamic multi-graph, since two nodes may have multiple temporal edges between if they collaborate over multiple years.

Splitting. We use the realistic training, validation and test edge splits provided by OGB. Specifically, they use collaborations until 2017 as training edges, those in 2018 as validation edges, and those in 2019 as test edges.

Training. All GNNs use a consistent learning rate strategy: an initial learning rate is set to 1×10^{-3}, the reduce factor is 0.5, the patience value is 10, and the stopping learning rate is 1×10^{-5}.

Performance Measure. We use the evaluator provided by OGB, which aims to measure a model's ability to predict future collaboration relationships given past collaborations. Specifically, they rank each true collaboration among a set of 100,000 randomly-sampled negative collaborations, and count the ratio of positive edges that are ranked at K-place or above (Hits@K, with $K = 50$).

Matrix Factorization Baseline. In addition to GNNs, we report performance for a simple matrix factorization baseline (Hu et al., 2020), which trains 256-dimensional embeddings for

		WikiCS				
Model	*L*	**#Param**	**Test Acc.±s.d.**	**Train Acc.±s.d.**	**#Epoch**	**Epoch/Total**
MLP	4	110710	59.452±2.327	85.347±5.440	322.46	0.01s/0.03hr
vanilla GCN	4	104560	77.103±0.830	98.918±0.619	293.84	0.05s/0.10hr
GraphSage	4	101775	74.767±0.950	99.976±0.095	303.68	0.06s/0.12hr
GCN	4	104560	77.469±0.854	98.925±0.590	299.85	0.06s/0.11hr
MoNet	4	106182	77.431±0.669	98.737±0.710	355.81	0.17s/0.36hr
MoNet-PE	4	107862	**77.481±0.712**	98.767±0.726	357.74	0.19s/0.81hr
GAT	4	105520	76.908±0.821	99.914±0.262	275.48	1.12s/2.22hr
GatedGCN	4	109280	OOM			
GIN	4	109782	75.857±0.577	99.575±0.388	321.25	0.06s/0.13hr

Table 6: Benchmarking results for WikiCS for node classification. Results (higher is better) are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models. The suffix -PE denote the use of Laplacian Eigenvectors as node positional encodings with dimension 20.

each of the 235K nodes. Comparing GNNs to matrix factorization tells us whether models leverage node features in addition to graph structure, as matrix factorization can be thought of as feature-agnostic.

Results. The numerical results are presented in Table 5 and discussed in Section D.

C.4 Node Classification with WikiCS dataset

WikiCS is a node classification dataset based on an extracted subset of Wikipedia's Computer Science articles (Mernyei and Cangea, 2020). It is a single graph dataset with $11,701$ nodes and $216,123$ edges where each node corresponds to an article, and each edge corresponds to a hyperlink. Each node belongs to a label out of total 10 classes representing the article's category. The average of the article text's pre-trained GloVe word embeddings (Pennington et al., 2014) is assigned as 300-dimensional node features. Compared to previous single-graph node classification benchmarks such as Cora and Citeseer, WikiCS dataset has denser node neighborhoods and each node's connectivity is spread across nodes from varying class labels. Additionally, as shown in Mernyei and Cangea (2020), the average shortest path length in WikiCS is smaller compared to Cora and Citeseer. Thus, on average, a larger node neighborhood and smaller shortest path length makes WikiCS an appropriate benchmark to test out neighborhood computation functions in GNNs' design.

Splitting. We follow the splitting defined in Mernyei and Cangea (2020) that has 20 different training, validation and early stopping splits consisting of 5% nodes, 22.5% nodes and 22.5% nodes of each class respectively. 50% nodes from each class, which are not in the training or validation split, are assigned as test splits. We combine the two original validation (22.5% nodes) and early stopping (22.5% nodes) splits to make the new validation (45% nodes) splits since we do not use separate early stopping splits in our benchmark.

Training. As consistent learning rate strategy across GNNs, an initial learning rate is set to 1×10^{-2}, the reduce factor is 0.5, the patience value is 25, and the stopping learning rate is 1×10^{-5}. Since there are 20 different training and validation splits, the training is done 20 times using these splits, and evaluated on the single test split. This is done for 4 times with 4 different seeds. Finally, the average of the $20 \times 4 = 80$ runs is reported.

Performance Measure. The performance measure is the classification accuracy between the predicted and groundtruth label for each node.
Results. The numerical results are presented in Table 6 and discussed in Section D.

C.5 Graph Classification with Super-pixel (MNIST/CIFAR10) datasets

The super-pixels datasets test graph classification using the popular MNIST and CIFAR10 image classification datasets. Our main motivation to use these datasets is as sanity-checks: we expect most GNNs to perform close to 100% for MNIST and well enough for CIFAR10. Besides, the use of super-pixel image datasets is suggestive of the way image datasets can be used for graph learning research.

The original MNIST and CIFAR10 images are converted to graphs using super-pixels. Super-pixels represent small regions of homogeneous intensity in images, and can be extracted with the SLIC technique (Achanta et al., 2012). We use SLIC super-pixels from (Knyazev et al., 2019)[2]. For each sample, we build a k-nearest neighbor adjacency matrix with

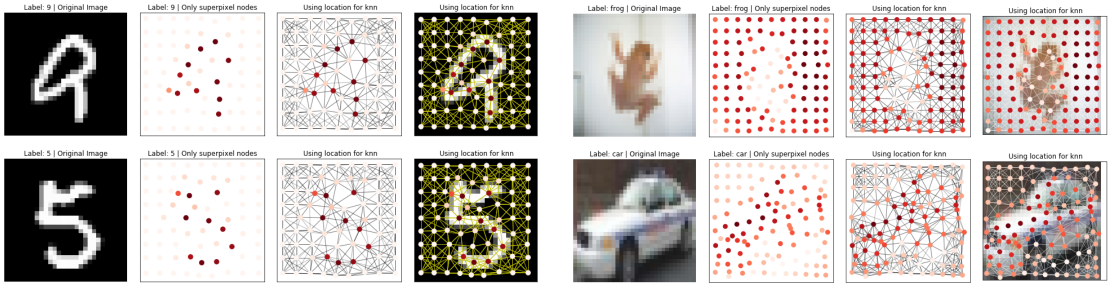
$$W_{ij}^{k-\mathrm{NN}} = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma_x^2}\right), \tag{47}$$

where x_i, x_j are the 2-D coordinates of super-pixels i, j, and σ_x is the scale parameter defined as the averaged distance x_k of the k nearest neighbors for each node. We use $k = 8$ for both MNIST and CIFAR10, whereas the maximum number of super-pixels (nodes) are 75 and 150 for MNIST and CIFAR10, respectively. The resultant graphs are of sizes 40-75 nodes for MNIST and 85-150 nodes for CIFAR10. Figure 14 presents visualizations of the super-pixel graphs.
Splitting. We use the standard splits of MNIST and CIFAR10. MNIST has $55,000$ train, $5,000$ validation, $10,000$ test graphs and CIFAR10 has $45,000$ train, $5,000$ validation, $10,000$ test graphs. The $5,000$ graphs for validation set are randomly sampled from the training set and the same splits are used for every GNN.
Training. The learning decay rate strategy is adopted with an initial learning rate of 1×10^{-3}, reduce factor 0.5, patience value 10, and the stopping learning rate 1×10^{-5} for all GNNs, except for 3WLGNN and RingGNN where we experienced a difficulty in training, leading us to slightly adjust their learning rate schedule hyperparameters. For both 3WLGNN and RingGNN, the patience value is changed to 5. For RingGNN, the initial learning rate is changed to 1×10^{-4} and the stopping learning rate is changed to 1×10^{-6}.
Performance Measure. The classification accuracy between the predicted and groundtruth label for each graph is the performance measure.
Results. The numerical results are presented in Table 7 and discussed in Section D.

C.6 Node Classification with SBM (PATTERN/CLUSTER) datasets

The SBM datasets consider node-level tasks of graph pattern recognition (Scarselli et al., 2009) – PATTERN and semi-supervised graph clustering – CLUSTER. The graphs are generated with the Stochastic Block Model (SBM) (Abbe, 2017), which is widely used to model communities in social networks by modulating the intra- and extra-communities

[2] `https://github.com/bknyaz/graph_attention_pool`


Label: 9 | Original Image
Label: 9 | Only superpixel nodes
Using location for knn
Using location for knn
Label: 5 | Original Image
Label: 5 | Only superpixel nodes
Using location for knn
Using location for knn
Label: frog | Original Image
Label: frog | Only superpixel nodes
Using location for knn
Using location for knn
Label: car | Original Image
Label: car | Only superpixel nodes
Using location for knn
Using location for knn

(a) MNIST

(b) CIFAR10

Figure 14: Sample images and their superpixel graphs. The graphs of SLIC superpixels (at most 75 nodes for MNIST and 150 nodes for CIFAR10) are 8-nearest neighbor graphs in the Euclidean space and node colors denote the mean pixel intensities.

		MNIST					CIFAR10				
Model	L	**#Param**	**Test Acc.±s.d.**	**Train Acc.±s.d.**	**#Epoch**	**Epoch/Total**	**#Param**	**Test Acc.±s.d.**	**Train Acc.±s.d.**	**#Epoch**	**Epoch/Total**
MLP	4	104044	95.340±0.138	97.432±0.470	232.25	22.74s/1.48hr	104380	56.340±0.181	65.113±1.685	185.25	29.48s/1.53hr
vanilla GCN	4	101365	90.705±0.218	97.196±0.223	127.50	83.41s/2.99hr	101657	55.710±0.381	69.523±1.948	142.50	109.70s/4.39hr
GraphSage	4	104337	**97.312±0.097**	100.000±0.000	98.25	113.12s/3.13hr	104517	65.767±0.308	99.719±0.062	93.50	124.61s/3.29hr
GCN	4	101365	90.120±0.145	96.459±1.020	116.75	37.06s/1.22hr	101657	54.142±0.394	70.163±3.429	140.50	47.16s/1.86hr
MoNet	4	104049	90.805±0.032	96.609±0.440	146.25	93.19s/3.82hr	104229	54.655±0.518	65.911±2.515	141.50	97.13s/3.85hr
GAT	4	110400	95.535±0.205	99.994±0.008	104.75	42.26s/1.25hr	110704	64.223±0.455	89.114±0.499	103.75	55.27s/1.62hr
GatedGCN	4	104217	**97.340±0.143**	100.000±0.000	96.25	128.79s/3.50hr	104357	**67.312±0.311**	94.553±1.018	97.00	154.15s/4.22hr
GIN	4	105434	96.485±0.252	100.000±0.000	128.00	39.22s/1.41hr	105654	55.255±1.527	79.412±9.700	141.50	52.12s/2.07hr
RingGNN	2	105398	11.350±0.000	11.235±0.000	14.00	2945.69s/12.77hr	105165	19.300±16.108	19.556±16.397	13.50	3112.96s/13.00hr
	2	505182	91.860±0.449	92.169±0.505	16.25	2575.99s/12.63hr	504949	39.165±17.114	40.209±17.790	13.75	2998.24s/12.60hr
	8	506357	Diverged	Diverged	Diverged	Diverged	510439	Diverged	Diverged	Diverged	Diverged
3WLGNN	3	108024	95.075±0.961	95.830±1.338	27.75	1523.20s/12.40hr	108516	59.175±1.593	63.751±2.697	28.50	1506.29s/12.60hr
	3	501690	95.002±0.419	95.692±0.677	26.25	1608.73s/12.42hr	502770	58.043±2.512	61.574±3.575	20.00	2091.22s/12.55hr
	8	500816	Diverged	Diverged	Diverged	Diverged	501584	Diverged	Diverged	Diverged	Diverged

Table 7: Benchmarking results for Super-pixels datasets for graph classification. Results (higher is better) are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models.

connections, thereby controlling the difficulty of the task. A SBM is a random graph which assigns communities to each node as follows: any two vertices are connected with the probability p if they belong to the same community, or they are connected with the probability q if they belong to different communities (the value of q acts as the noise level).

PATTERN: The graph pattern recognition task, presented in Scarselli et al. (2009), aims at finding a fixed graph pattern P embedded in larger graphs G of variable sizes. For all data, we generate graphs G with 5 communities with sizes randomly selected between $[5, 35]$. The SBM of each community is $p = 0.5, q = 0.35$, and the node features on G are generated with a uniform random distribution with a vocabulary of size 3, *i.e.* $\{0, 1, 2\}$. We randomly generate 100 patterns P composed of 20 nodes with intra-probability $p_P = 0.5$ and extra-probability $q_P = 0.5$ (*i.e.*, 50% of nodes in P are connected to G). The node features for P are also generated as a random signal with values $\{0, 1, 2\}$. The graphs are of sizes 44-188 nodes. The output node labels have value 1 if the node belongs to P and value 0 if it is in G.

CLUSTER: For the semi-supervised clustering task, we generate 6 SBM clusters with sizes randomly selected between $[5, 35]$ and probabilities $p = 0.55, q = 0.25$. The graphs are of sizes 40-190 nodes. Each node can take an input feature value in $\{0, 1, 2, .., 6\}$. If the value

		PATTERN					CLUSTER				
Model	L	**#Param**	**Test Acc.±s.d.**	**Train Acc.±s.d.**	**#Epoch**	**Epoch/Total**	**#Param**	**Test Acc.±s.d.**	**Train Acc.±s.d.**	**#Epoch**	**Epoch/Total**
MLP	4	105263	50.519±0.000	50.487±0.014	42.25	8.95s/0.11hr	106015	20.973±0.004	20.938±0.002	42.25	5.83s/0.07hr
vanilla GCN	4	100923	63.880±0.074	65.126±0.135	105.00	118.85s/3.51hr	101655	53.445±2.029	54.041±2.197	70.00	65.72s/1.30hr
	16	500823	71.892±0.334	78.409±1.592	81.50	492.19s/11.31hr	501687	68.498±0.976	71.729±2.212	79.75	270.28s/6.08hr
GraphSage	4	101739	50.516±0.001	50.473±0.014	43.75	93.41s/1.17hr	102187	50.454±0.145	54.374±0.203	64.00	53.56s/0.97hr
	16	502842	50.492±0.001	50.487±0.005	46.50	391.19s/5.19hr	503350	63.844±0.110	86.710±0.167	57.75	225.61s/3.70hr
GCN	4	100923	85.498±0.045	85.598±0.043	65.00	19.21s/0.36hr	101655	47.828±1.510	48.258±1.607	63.50	12.84s/0.23hr
	16	500823	85.614±0.032	86.034±0.087	66.00	37.08s/0.70hr	501687	69.026±1.372	73.749±2.570	77.75	30.20s/0.66hr
MoNet	4	103775	85.482±0.037	85.569±0.044	89.75	35.71s/0.90hr	104227	58.064±0.131	58.454±0.183	76.25	24.29s/0.52hr
	16	511487	85.582±0.038	85.720±0.068	81.75	68.49s/1.58hr	511999	66.407±0.540	67.727±0.649	77.75	47.82s/1.05hr
GAT	4	109936	75.824±1.823	77.883±1.632	96.00	20.92s/0.57hr	110700	57.732±0.323	58.331±0.342	67.25	14.17s/0.27hr
	16	526990	78.271±0.186	90.212±0.476	53.50	50.33s/0.77hr	527874	70.587±0.447	76.074±1.362	73.50	35.94s/0.75hr
GatedGCN	4	104003	84.480±0.122	84.474±0.155	78.75	139.01s/3.09hr	104355	60.404±0.419	61.618±0.536	94.50	79.97s/2.13hr
	16	502223	85.568±0.088	86.007±0.123	65.25	644.71s/11.91hr	502615	73.840±0.326	87.880±0.908	60.00	400.07s/6.81hr
GatedGCN-PE	16	502457	**86.508±0.085**	86.801±0.133	65.75	647.94s/12.08hr	504253	**76.082±0.196**	88.919±0.720	57.75	399.66s/6.58hr
GIN	4	100884	85.590±0.011	85.852±0.030	93.00	15.24s/0.40hr	103544	58.384±0.236	59.480±0.337	74.75	10.71s/0.23hr
	16	508574	85.387±0.136	85.664±0.116	86.75	25.14s/0.62hr	517570	64.716±1.553	65.973±1.816	80.75	20.67s/0.47hr
RingGNN	2	105206	86.245±0.013	86.118±0.034	75.00	573.37s/12.17hr	104746	42.418±20.063	42.520±20.212	74.50	428.24s/8.79hr
	2	504766	86.244±0.025	86.105±0.021	72.00	595.97s/12.15hr	524202	22.340±0.000	22.304±0.000	43.25	501.84s/6.22hr
	8	505749	Diverged	Diverged	Diverged	Diverged	514380	Diverged	Diverged	Diverged	Diverged
3WLGNN	3	103572	85.661±0.353	85.608±0.337	95.00	304.79s/7.88hr	105552	57.130±6.539	57.404±6.597	116.00	219.51s/6.52hr
	3	502872	85.341±0.207	85.270±0.198	81.75	424.23s/9.56hr	507252	55.489±7.863	55.736±8.024	66.00	319.98s/5.79hr
	8	581716	Diverged	Diverged	Diverged	Diverged	586788	Diverged	Diverged	Diverged	Diverged

Table 8: Benchmarking results for SBMs datasets for node classification. Results (higher is better) are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models. The suffix -PE denote the use of Laplacian Eigenvectors as node positional encodings with dimension 2 for PATTERN and 20 for CLUSTER.

is 1, the node belongs to class 0, value 2 corresponds to class 1, . . . , value 6 corresponds to class 5. Otherwise, if the value is 0, the class of the node is unknown and will be inferred by the GNN. There is only one labelled node that is randomly assigned to each community and most node features are set to 0. The output node labels are defined as the community/cluster class labels.

Splitting. The PATTERN dataset has $10,000$ train, $2,000$ validation, $2,000$ test graphs and CLUSTER dataset has $10,000$ train, $1,000$ validation, $1,000$ test graphs. We save the generated splits and use the same sets in all models for fair comparison.

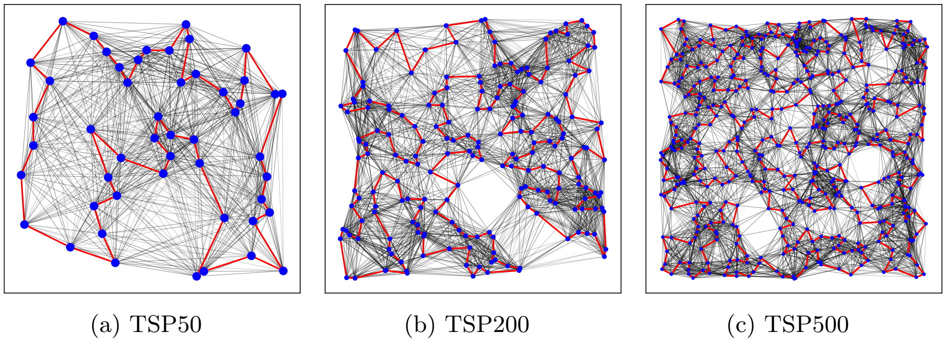
Training. As presented in the standard experimental protocol in Section C, we use Adam optimizer with a learning rate decay strategy. For all GNNs, an initial learning rate is set to 1×10^{-3}, the reduce factor is 0.5, the patience value is 5, and the stopping learning rate is 1×10^{-5}.

Performance Measure. The performance measure is the average node-level accuracy weighted with respect to the class sizes.

Results. Our numerical results are presented in Table 8 and discussed in Section D together with other benchmark results.

C.7 Edge Classification/Link Prediction with TSP dataset

Leveraging machine learning for solving NP-hard combinatorial optimization problems (COPs) has been the focus of intense research in recent years (Vinyals et al., 2015; Bengio et al., 2018). Recently proposed learning-driven solvers for COPs (Khalil et al., 2017; Kool et al., 2019; Joshi et al., 2019) combine GNNs with classical search to predict approximate solutions directly from problem instances (represented as graphs). Consider the intensively studied Travelling Salesman Problem (TSP), which asks the following question: *"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits*



(a) TSP50 (b) TSP200 (c) TSP500

Figure 15: Sample graphs from the TSP dataset. Nodes are colored blue and edges on the groundtruth TSP tours are colored red.

each city and returns to the origin city?" Formally, given a 2D Euclidean graph, one needs to find an optimal sequence of nodes, called a tour, with minimal total edge weights (tour length). TSP's *multi-scale* nature makes it a challenging graph task which requires reasoning about both local node neighborhoods as well as global graph structure.

For our experiments with TSP, we follow the learning-based approach to COPs described in Joshi et al. (2022), where a GNN is the backbone architecture for assigning probabilities to each edge as belonging/not belonging to the predicted solution set. The probabilities are then converted into discrete decisions through graph search techniques. Each instance is a graph of n node locations sampled uniformly in the unit square $S = \{x_i\}_{i=1}^n$ and $x_i \in [0, 1]^2$. We generate problems of varying size and complexity by uniformly sampling the number of nodes $n \in [50, 500]$ for each instance.

In order to isolate the impact of the backbone GNN architectures from the search component, we pose TSP as a binary edge classification task, with the groundtruth value for each edge belonging to the TSP tour given by Concorde (Applegate et al., 2006). For scaling to large instances, we use sparse $k = 25$ nearest neighbor graphs instead of full graphs, following (Khalil et al., 2017). See Figure 15 for sample TSP instances of various sizes.

Splitting. TSP has $10,000$ train, $1,000$ validation and $1,000$ test graphs.
Training. All GNNs use a consistent learning rate strategy: an initial learning rate is set to 1×10^{-3}, the reduce factor is 0.5, the patience value is 10, and the stopping learning rate is 1×10^{-5}.
Performance Measure. Given the high class imbalance, *i.e.*, only the edges in the TSP tour have positive label, we use the F1 score for the positive class as our performance measure.
Non-learnt Baseline. In addition to reporting performance of GNNs, we compare with a simple k-nearest neighbor heuristic baseline, defined as follows: Predict true for the edges corresponding to the k nearest neighbors of each node, and false for all other edges. We set $k = 2$ for optimal performance. Comparing GNNs to the non-learnt baseline tells us whether models learn something more sophisticated than identifying a node's nearest neighbors.
Results. The numerical results are presented in Table 9 and analysed in Section D.

				TSP		
Model	L	**#Param**	**Test F1±s.d.**	**Train F1±s.d.**	**#Epoch**	**Epoch/Total**
MLP	4	96956	0.544±0.001	0.544±0.001	164.25	50.15s/2.31hr
vanilla GCN	4	95702	0.630±0.001	0.631±0.001	261.00	152.89s/11.15hr
GraphSage	4	99263	0.665±0.003	0.669±0.003	266.00	157.26s/11.68hr
GCN	4	95702	0.643±0.001	0.645±0.002	261.67	57.84s/4.23hr
MoNet	4	99007	0.641±0.002	0.643±0.002	282.00	84.46s/6.65hr
GAT	4	96182	0.671±0.002	0.673±0.002	328.25	68.23s/6.25hr
GatedGCN	4	97858	0.791±0.003	0.793±0.003	159.00	218.20s/9.72hr
GatedGCN-E	4	97858	0.808±0.003	0.811±0.003	197.00	218.51s/12.04hr
GatedGCN-E	16	500770	**0.838±0.002**	0.850±0.001	53.00	807.23s/12.17hr
GIN	4	99002	0.656±0.003	0.660±0.003	273.50	72.73s/5.56hr
RingGNN	2	106862	0.643±0.024	0.644±0.024	2.00	17850.52s/17.19hr
	2	507938	0.704±0.003	0.705±0.003	3.00	12835.53s/16.08hr
	8	506564	Diverged	Diverged	Diverged	Diverged
3WLGNN	3	106366	0.694±0.073	0.695±0.073	2.00	17468.81s/16.59hr
	3	506681	0.288±0.311	0.290±0.312	2.00	17190.17s/16.51hr
	8	508832	OOM	OOM	OOM	OOM
k-NN Heuristic		k =2	Test F1: 0.693			

Table 9: Benchmarking results for TSP for edge classification. Results (higher is better) are averaged over 4 runs with 4 different seeds. **Red**: the best model, Violet: good models. The suffix -E denotes the use of available edge features.

C.8 Graph Classification and Isomorphism Testing with CSL dataset

The Circular Skip Link dataset is a symmetric graph dataset introduced in Murphy et al. (2019) to test the expressivity of GNNs. Each CSL graph is a 4-regular graph with edges connected to form a cycle and containing skip-links between nodes. Formally, it is denoted by $\mathcal{G}_{N,C}$ where N is the number of nodes and C is the isomorphism class which is the skip-link length of the graph. We use the same dataset $\mathcal{G}_{41,C}$ with $C \in \{2, 3, 4, 5, 6, 9, 11, 12, 13, 16\}$. The dataset is class-balanced with 15 graphs for every C resulting in a total of 150 graphs.

Splitting. We perform a 5-fold cross validation split, following Murphy et al. (2019), which gives 5 sets of train, validation and test data indices in the ratio $3 : 1 : 1$. We use stratified sampling to ensure that the class distribution remains the same across splits. The indices are saved and used across all experiments for fair comparisons.

Training. For the learning rate strategy across all GNNs, an initial learning rate is set to 5×10^{-4}, the reduce factor is 0.5, the patience value is 5, and the stopping learning rate is 1×10^{-6}. We train on the 5-fold cross validation with 20 different seeds of initialization, following Chen et al. (2019b).

Performance Measure. We use graph classification accuracy between the predicted labels and groundtruth labels as our performance measure. The model performance is evaluated on the test split of the 5 folds at every run, and following Murphy et al. (2019); Chen et al. (2019b), we report the maximum, minimum, average and the standard deviation of the 100 scores, *i.e.*, 20 runs of 5-folds.

Results. The numerical results are reported in Table 10 and analyzed in Section E.1. In this paper, we use CSL primarily to validate the impact of having Graph Positional Encodings

Model	L	#Param	Test Accuracy			Train Accuracy			#Epoch	Epoch/ Total
			Mean±s.d.	Max	Min	Mean±s.d.	Max	Min		
Node Positional Encoding with Laplacian Eigenvectors										
MLP	4	101235	22.567±6.089	46.667	10.000	30.389±5.712	43.333	10.000	109.39	0.16s/0.03hr
GCN	4	103847	**100.000±0.000**	100.000	100.000	100.000±0.000	100.000	100.000	125.64	0.40s/0.07hr
GraphSage	4	105867	99.933±0.467	100.000	96.667	100.000±0.000	100.000	100.000	155.00	0.50s/0.11hr
MoNet	4	105579	99.967±0.332	100.000	96.667	100.000±0.000	100.000	100.000	130.39	0.49s/0.09hr
GAT	4	101710	99.933±0.467	100.000	96.667	100.000±0.000	100.000	100.000	133.18	0.61s/0.12hr
GatedGCN	4	105407	99.600±1.083	100.000	96.667	100.000±0.000	100.000	100.000	147.06	0.66s/0.14hr
GIN	4	107304	99.333±1.333	100.000	96.667	100.000±0.000	100.000	100.000	62.98	0.44s/0.04hr
RingGNN	2	102726	17.233±6.326	40.000	10.000	26.122±14.382	58.889	10.000	122.75	2.93s/0.50hr
	2	505086	25.167±7.399	46.667	10.000	54.533±18.415	82.222	10.000	120.58	3.11s/0.51hr
3WLGNN	3	102054	30.533±9.863	56.667	10.000	99.644±1.684	100.000	88.889	74.66	2.33s/0.25hr
	3	505347	30.500±8.197	56.667	13.333	100.000±0.000	100.000	100.000	66.64	2.38s/0.23hr
No Node Positional Encoding										
All MP-GCNs	4	100K	10.000±0.000	10.000	10.000	10.000±0.000	10.000	10.000	-	-
RingGNN	2	101138	10.000±0.000	10.000	10.000	10.000±0.000	10.000	10.000	103.23	3.09s/0.45hr
	2	505325	10.000±0.000	10.000	10.000	10.000±0.000	10.000	10.000	90.04	3.28s/0.42hr
3WLGNN	3	102510	95.700±14.850	100.000	30.000	95.700±14.850	100.000	30.000	475.81	2.29s/1.51hr
	3	506106	97.800±10.916	100.000	30.000	97.800±10.916	100.000	30.000	283.80	2.28s/0.90hr

Table 10: Results for the CSL dataset, with and without Laplacian Positional Encodings. Results are from 5-fold cross validation, run 20 times with different seeds. **Red**: the best model, Violet: good models. The dimension of node positional encoding with Laplacian eigenvectors is 20.

(Section E.1) that is proposed as a demonstration of our benchmarking framework to steer new GNN research.

C.9 Cycle Detection with CYCLES dataset

The CYCLES is a dataset synthetically generated by Loukas (2020) which contains equal number of graphs with and without cycles of fixed lengths. The task is a binary classification task to detect whether a graph has cycle or not. Though there are several forms of the dataset used in Loukas (2020) in terms of the number of nodes and cycle lengths, we select the dataset variant marked with having node size 56 and cycle length 6, based on the difficulty results shown by the author. The graphs have nodes in the range 37-65.

Splitting. We use the same dataset splits as in Loukas (2020). Originally there 10,000 graphs each in the training and test sets. We sample 1,000 class balanced graphs from the training set to be used as validation samples. Therefore, the resulting CYCLES dataset has 9,000 train/ 1,000 validation/10,000 test graphs with all the sets having class-balanced samples. We show results on different sizes of training samples following the original author of CYCLES dataset.

Training. For the learning rate strategy, an initial learning rate is set to 1×10^{-4}, the reduce factor is 0.5, the patience value is 10, and the stopping learning rate is 1×10^{-6}. Following Loukas (2020), we train using a varying sample size from 200 to $5,000$ out of the training graphs and report the results accordingly. The reported results are based on 4 runs with 4 different seeds.

Performance Measure. The classification accuracy between the predicted and groundtruth label for whether a graph has cycle or not is the performance measure.

Results. Similar to the CSL dataset (Section C.8), we use the CYCLES dataset mainly for the validation of the Graph Positional Encodings (Section E.1) proposed as an outcome of this benchmarking framework. As such, we train only a subset of MP-GCNs (GINs and GatedGCNs) and report the respective results. The numerical results are reported in Table 11 and analyzed in Section E.1.

		Train samples →	**200**	**500**	**1000**	**5000**
Model	L	**#Param**	**Test Acc±s.d.**			
GIN	4	100774	70.585±0.636	74.995±1.226	78.083±1.083	86.130±1.140
GIN-PE	4	102864	**86.720±3.376**	**95.960±0.393**	**97.998±0.300**	**99.570±0.089**
GatedGCN	4	103933	50.000±0.000	50.000±0.000	50.000±0.000	50.000±0.000
GatedGCN-PE	4	105263	**95.082±0.346**	**96.700±0.381**	**98.230±0.473**	**99.725±0.027**

Table 11: Test accuracy on the CYCLES dataset. Results (higher is better) are averaged over 4 runs with 4 different seeds. The performance on test sets with models trained on varying train data size is show, following Vignac et al. (2020). **Bold** shows the best result out of a GNN's two model instances that use and not use PE. The dimension for PE is 20.

C.10 Multi-task graph properties with GraphTheoryProp dataset

Corso et al. (2020) proposed a synthetic dataset of undirected and unweighted graphs of diverse types randomly generated for a multi-task benchmarking of 6 graph-theoretic properties, 3 at the node-level and 3 at the graph-level. We call this dataset as GraphTheoryProp. The node-level tasks are to determine single source shortest paths (Dist.), node eccentricity (Ecc.), and Laplacian features LX given a node feature vector X (Lap.) The graph-level tasks are graph connectivity (Conn.), diameter (Diam.) and spectral radius (Rad.). The dataset has graph sizes in the range of 15-24 nodes which have random identifiers as input features. This dataset is crucial to benchmark the robustness of a GNN to predict specific or overall of all the 6 properties, as these may share subroutines such as graph traversals, despite the tasks being different graph properties (Corso et al., 2020).

Model	L	**Test**						
		Average	**Dist.**	**Ecc.**	**Lap.**	**Conn.**	**Diam.**	**Rad.**
GIN	8	-3.19±0.11	-2.81±0.11	-2.42±0.09	**-4.39±0.18**	**-2.07±0.13**	-3.06±0.11	**-4.39±0.13**
GIN-PE	8	**-3.21±0.13**	**-2.87±0.03**	**-2.83±0.07**	-3.99±0.04	-2.00±0.15	**-3.27±0.07**	-4.31±0.15
GatedGCN	8	-3.22±0.13	-2.76±0.17	-2.36±0.12	-3.92±0.15	**-2.65±0.11**	-3.35±0.16	-4.31±0.08
GatedGCN-PE	8	**-3.51±0.11**	**-3.23±0.08**	**-3.35±0.08**	**-4.03±0.21**	-2.60±0.12	**-3.57±0.05**	-4.32±0.13

Table 12: Mean Log_{10}MSE for each task over 4 runs with 4 different seeds. Average denotes the combined average of all the tasks. Log_{10}MSE is on the test set (lower is better). **Bold** shows the best result out of a GNN's two model instances that use and not use PE. The dimension for PE is 12.

Splitting. We use the same splitting sets as in Corso et al. (2020) which has 5,120 train, 640 validation, 1,280 test graphs.

Training. For the learning rate strategy, an initial learning rate is set to 1×10^{-3}, the reduce factor is 0.5, the patience value is 15, and the stopping learning rate is 1×10^{-6}. The reported results are based on 4 runs with 4 different seeds.

Performance Measure. For performance measure, Log_{10}MSE is reported between the

predicted and groundtruth values for each single task. Besides, an average performance measure is reported which is the combined average of all the 6 tasks.
Results. As with the CSL and CYCLES datasets (Sections C.8, C.9), we use GraphTheoryProp in this paper for the validation of Graph Positional Encodings, Section E.1. The numerical results are reported in Table 12 and analyzed in Section E.1.

Appendix D. Analysis and Discussion of Benchmarking Results

This section highlights the main take-home messages from the experiments in Section C on the datasets in the proposed framework, which evaluate the GNNs from Section B with the experimental setup described in Section C and respective sub-sections of each datasets.

Graph-agnostic NNs perform poorly. As a sanity check, we compare all GNNs to a simple graph-agnostic MLP baseline which updates each node independent of one-other, $h_i^{\ell+1} = \sigma\left(W^\ell\ h_i^\ell\right)$, and passes these features to the task-based layer. MLP presents consistently low scores across all datasets (Tables 3-10), which shows the necessity to use graph structure for these tasks. All proposed datasets used in our study are appropriate to statistically separate GNN performance, which has remained an issue with the widely used but small graph datasets (Errica et al., 2019; Luzhnica et al., 2019).

GCNs outperform WL-GNNs on the proposed datasets. Although provably powerful in terms of graph isomorphism tests and invariant function approximation (Maron et al., 2019c; Chen et al., 2019b; Morris et al., 2019), the recent 3WLGNNs and RingGNNs were not able to outperform GCNs for our medium-scale datasets, as shown in Tables 3-5 and 7-9. These new models are limited in terms of space/time complexities, with $O(n^2)/O(n^3)$ respectively, not allowing them to scale to larger datasets. On the contrary, GCNs with linear complexity *w.r.t.* the number of nodes for sparse graphs, can scale conveniently to 16 layers and show the best performance on all datasets. 3WL-GNNs and RingGNNs face loss divergence and/or out-of-memory errors when trying to build deeper networks.

Anisotropic mechanisms improve GCNs. Among the models in the GCN class, the best results point towards the anisotropic models, particularly GAT and GatedGCN, which are based on sparse and dense attention mechanisms, respectively. For instance, results for ZINC, AQSOL, WikiCS, MNIST, CIFAR10, PATTERN and CLUSTER in respective Tables 3, 4, 6, 7, 8 show that the performance of the 100K-parameter anisotropic GNNs (GCN with symmetric normalization, GAT, MoNet, GatedGCN) are consistently better than the isotropic models (*vanilla* GCN, GraphSage), except for *vanilla* GCN-WikiCS, GraphSage-MNIST and MoNet-CIFAR10. Table 14, discussed later, dissects and demonstrates the importance of anisotropy for the link prediction tasks, TSP and COLLAB. Overall, our results suggest that understanding the expressive power of attention-based neighborhood aggregation functions is a meaningful avenue of research.

Underlying challenges for training WL-GNNs. We consistently observe a relatively high standard deviation in the performance of WL-GNNs (recall that we average across 4 runs using 4 different seeds). We attribute this fluctuation to the absence of universal training procedures like batching and batch normalization, as these GNNs operate on *dense* rank-2 tensors of variable sizes. On the other hand, GCNs running on *sparse* tensors better leverage batched training and normalization for stable and fast training. Leading graph machine learning libraries represent batches of graphs as sparse block diagonal matrices, enabling batched training of GCNs through parallelized computation (Jia et al., 2019).

Dense tensors are incompatible with the prevalent approach, disabling the use of batch normalization for WL-GNNs. We experimented with layer normalization (Ba et al., 2016) but without success. We were also unable to train WL-GNNs on CPU memory for the single COLLAB graph, see Table 5. Practical applications of the new WL-GNNs may require redesigning the best practices and common building blocks of deep learning, *i.e.* batching of variable-sized data, normalization schemes, and residual connections.

3WL-GNNs perform the best among their class. Among the models in the WL-GNN class, 3WL-GNN provide better results than its similar counter-part RingGNN and achieves close to the best performance for AQSOL, see Table 4. The GIN model, while being less expressive, is able to scale better and provides overall good performance.

Appendix E. Studies using the Benchmarking Framework

One of the primary goals of this benchmarking framework is to facilitate researchers to perform new explorations conveniently and develop insights that improve our overall understanding of graph neural networks. This section provides a demonstration of two such studies that we carry out by leveraging the datasets and the coding infrastructure which are part of this framework. First, we explore the absence of positional information in graphs for MP-GCNs which induces their low representation power. As a result, we develop a new insight that Laplacian eigenvectors can very simply be used as graph positional encodings and improve MP-GCNs. This insight has been received keenly in the recent literature and there are a number of works that propose positional encoding schemes with some addressing the challenges of using Laplacian eigenvectors (Kreuzer et al., 2021; Wang et al., 2022; Lim et al., 2022). Second, we study and show how the modification of existing MP-GCNs with joint edge representations help the models perform comparatively better than their vanilla counterparts.

E.1 Laplacian Positional Encodings

As discussed in Section D, MP-GCNs outperforms WL-GNNs on the diverse collection of datasets included in our proposed benchmark despite having theoretical limitations derived from the alignment of MP-GCNs to the WL-tests. Also, WL-GNNs were found to be computationally infeasible on medium and large scale datasets. Motivated by these results, we propose 'Graph Positional Encodings' using Laplacian eigenvectors, thus referred as Laplacian Positional Encodings, to improve the theoretical shortcomings of MP-GCNs, which allows us to retain the computationally efficiency offered by the message-passing framework and improve the MP-GCNs performance.

E.1.1 Related Work

In Murphy et al. (2019); Srinivasan and Ribeiro (2020), it was pointed out that standard MP-GCNs might perform poorly when dealing with graphs that exhibit some symmetries in their structures, such as node or edge isomorphism. This is related to the limitation of MP-GCNs due to their equivalence to the 1-WL test (Xu et al., 2019; Morris et al., 2019).

The equivalence is based on the condition when MP-GCNs handle anonymous nodes (Loukas, 2020), i.e. nodes do not have unique node features. To address this issue of anonymous MP-GCNs, Murphy et al. (2019) introduced a framework, called Graph Relational Pooling (GRP), that assigns to each node an identifier that depends on the index ordering. This approach can be computationally expensive as it requires to account for all $n!$ node permutations, thus requiring some sampling in practice. You et al. (2019) proposed learnable position-aware embeddings based on random anchor sets of nodes for pairwise node (or, link) tasks. However, the random selection of anchor sets has limitations and their approach is not applicable on inductive node tasks. Similarly, one could think of using full or partial random node identifiers for breaking node-anonymity. Yet, it suffers from generalization to unseen graphs (You et al., 2019; Loukas, 2020). Li et al. (2020) proposed the use of distance encoding as node attributes which captures distances between nodes using power(s) of random walk matrix. However, their failure on distance regular graphs (Li et al., 2020) and the cost of computing the power matrices may be limiting to scale to diverse and medium to large-scale graphs. We improve upon these works and propose the use of Laplacian eigenvectors as positional encodings.

E.1.2 Laplacian eigenvectors as Positional Encodings

We keep the overall MP-GCN architecture and simply add positional features to each node before processing the graph through the MP-GCN. Intuitively, the positional features should be chosen such that nodes which are far apart in the graph have different positional features whereas nodes which are nearby have similar positional features. As node positional features, we propose to use graph Laplacian eigenvectors (Belkin and Niyogi, 2003), which have less ambiguities and which better describe the distance between nodes on the graph. Formally, Laplacian eigenvectors are spectral techniques that embed the graphs into the Euclidean space. These vectors form a meaningful local coordinate system, while preserving the global graph structure. Mathematically, they are defined via the factorization of the graph Laplacian matrix;

$$\Delta = \mathrm{I} - D^{-1/2} A D^{-1/2} = U^T \Lambda U, \tag{48}$$

where A is the $n \times n$ adjacency matrix, D is the degree matrix, and Λ, U correspond respectively to the eigenvalues and eigenvectors. Laplacian eigenvectors also represent a natural generalization of the Transformer (Vaswani et al., 2017) positional encodings (PE) for graphs as the eigenvectors of a discrete line (NLP graph) are the cosine and sinusoidal functions. The computational complexity $O(E^{3/2})$, with E being the number of edges, can be improved with, *e.g.* the Nystrom method (Fowlkes et al., 2004). The eigenvectors are defined up to the factor ± 1 (after being normalized to unit length), so the sign of eigenvectors will be randomly flipped during training. For the experiments, we use the k smallest non-trivial eigenvectors, where the k value is given in the respective experiment tables in Section C as the dimensions of the PE. The smallest eigenvectors provide smooth encoding coordinates of neighboring nodes. See Section E.1.4 for additional discussion about positional encodings and the reasoning behind our decision to use random sign flipping.

E.1.3 Experiments and Analysis

We first use the mathematical graphs such as CSL, CYCLES and GraphTheoryProp included in our benchmark (Sections C.8-C.10) to validate the proposed Laplacian PE as simple augmentations in MP-GCNs to improve their performance on the datasets. On CSL dataset, Table 10 compares the MP-GCNs using the Laplacian eigenvectors as PE and the WL-GNNs. The MP-GCN models were the most accurate with 99% of mean accuracy, while 3WL-GNN obtained 97% and RingGNN 25% with our experimental setting. Similarly, in Table 11 for CYCLES dataset and Table 12 for GraphTheoryProp dataset, where we simply select 2 representative MP-GCNs (GINs and GatedGCNs), we observe a consistent improvement in the performance when GINs and GatedGCNs are augmented with Laplacian PE. This demonstrates the importance of positional features to successfully detect cycles in a graph, and also predict critical theoretical and geometric properties in a graph.

Next, we study ZINC, AQSOL, WikiCS, PATTERN, CLUSTER and COLLAB with PE (note that MNIST, CIFAR10 and TSP do not need PE as the nodes in these graphs already have features describing their positions in $\mathbb{R}^2$). We observe a boost of performance for ZINC, AQSOL and CLUSTER (it was expected as eigenvectors are good indicators of clusters (Von Luxburg, 2007)), an improvement for PATTERN, and statistically the same result for COLLAB, see the respective tables in Section C. This way, MP-GCNs can be augmented with Laplacian PE to overcome their limitations of not being able to detect simple graph symmetries. Additionally, PEs also boost the models' performance on real-world graph learning tasks.

E.1.4 Challenges with using Laplacian eigenvectors

Ideally, positional encodings (PEs) should be unique for each node, and nodes which are far apart in the graph should have different positional features whereas nodes which are nearby have similar positional features. Note that in a graph that has some symmetries, positional features cannot be assigned in a canonical way. For example, if node i and node j are structurally symmetric, and we have positional features $p_i = a$, $p_j = b$ that differentiate them, then it is also possible to arbitrary choose $p_i = b$, $p_j = a$ since i and j are completely symmetric by definition. In other words, the PE is always arbitrary up to the number of symmetries in the graph. As a consequence, the network will have to learn to deal with these ambiguities during training. The simplest possible positional encodings is to give an (arbitrary) ordering to the nodes, among $n!$ possible orderings. During training, the orderings are uniformly sampled from the $n!$ possible choices in order for the network to learn to be independent to these arbitrary choices (Murphy et al., 2019).

We propose an alternative to reduce the sampling space, and therefore the amount of ambiguities to be resolved by the network. Laplacian eigenvectors are hybrid positional and structural encodings, as they are invariant by node re-parametrization. However, they are also limited by natural symmetries such as the arbitrary sign of eigenvectors (after being normalized to have unit length). The number of possible sign flips is 2^k, where k is the number of eigenvectors. In practice we choose $k \ll n$, and therefore 2^k is much smaller $n!$ (the number of possible ordering of the nodes). During the training, the eigenvectors will be uniformly sampled at random between the 2^k possibilities. If we do not seek to learn the invariance w.r.t. all possible sign flips of eigenvectors, then we can remove the sign ambiguity

	PE type	L	#Param	Test Acc.±s.d.	Train Acc.±s.d.	#Epochs	Epoch/Total
CSL	No PE	4	104007	10.000±0.000	10.000±0.000	54.00	0.58s/0.05hr
	EigVecs-20	4	105407	68.633±7.143	99.811±0.232	107.16	0.59s/0.09hr
	Rand sign(EigVecs)	4	105407	**99.767±0.394**	99.689±0.550	188.76	0.59s/0.16hr
	Abs(EigVecs)	4	105407	99.433±1.133	100.000±0.000	143.64	0.60s/0.12hr
	Fixed node ordering	4	106807	10.533±4.469	76.056±14.136	60.56	0.59s/0.05hr
	Rand node ordering	4	106807	11.133±2.571	10.944±2.106	91.60	0.60s/0.08hr
PATTERN	No PE	16	502223	85.605±0.105	85.999±0.145	62.00	646.03s/11.36hr
	EigVecs-2	16	505421	86.029±0.085	86.955±0.227	65.00	645.36s/11.94hr
	Rand sign(EigVecs)	16	502457	**86.508±0.085**	86.801±0.133	65.75	647.94s/12.08hr
	Abs(EigVecs)	16	505421	86.393±0.037	87.011±0.172	62.00	645.90s/11.41hr
	Fixed node ordering	16	516887	80.133±0.202	98.416±0.141	45.00	643.23s/8.27hr
	Rand node ordering	16	516887	85.767±0.044	85.998±0.063	64.50	645.09s/11.79hr
CLUSTER	No PE	16	502615	73.684±0.348	88.356±1.577	61.50	399.44s/6.97hr
	EigVecs-20	16	504253	75.520±0.395	89.332±1.297	49.75	400.50s/5.70hr
	Rand sign(EigVecs)	16	504253	**76.082±0.196**	88.919±0.720	57.75	399.66s/6.58hr
	Abs(EigVecs)	16	504253	73.796±0.234	91.125±1.248	58.75	398.97s/6.68hr
	Fixed node ordering	16	517435	69.232±0.265	92.298±0.712	51.00	400.40s/5.82hr
	Rand node ordering	16	517435	74.656±0.314	82.940±1.718	61.00	397.75s/6.88hr
COLLAB	No PE	3	40965	52.635±1.168	96.103±1.876	95.00	453.47s/12.09hr
	EigVecs-20	3	41889	52.326±0.678	96.700±1.296	95.00	452.40s/12.10hr
	Rand sign(EigVecs)	3	41889	**52.849±1.345**	96.165±0.453	94.75	452.75s/12.08hr
	Abs(EigVecs)	3	41889	51.419±1.109	95.984±1.157	95.00	451.36s/12.07hr
	PE type	L	**#Param**	**Test MAE±s.d.**	**Train MAE±s.d.**	**#Epochs**	**Epoch/Total**
ZINC	No PE	16	504153	0.354±0.012	0.095±0.012	165.25	10.52s/0.49hr
	EigVecs-8	16	505011	0.319±0.010	0.038±0.007	143.25	10.62s/0.43hr
	Rand sign(EigVecs)	16	505011	**0.214±0.013**	0.067±0.019	185.00	10.70s/0.56hr
	Abs(EigVecs)	16	505011	**0.214±0.009**	0.035±0.011	167.50	10.61s/0.50hr
	Fixed node ordering	16	507195	0.431±0.007	0.044±0.009	118.25	10.62s/0.35hr
	Rand node ordering	16	507195	0.321±0.015	0.177±0.015	184.75	10.55s/0.55hr

Table 13: Study of positional encodings (PEs) with the GatedGCN model (Bresson and Laurent, 2017). Performance reported on the test sets of CSL, ZINC, PATTERN, CLUSTER and COLLAB (higher is better, except for ZINC). **Red**: the best model.

of eigenvectors by taking the absolute value. This choice seriously degrades the expressivity power of the positional features.

Numerical results for different positional encodings are reported in Table 13. For all results, we use the GatedGCN model (Bresson and Laurent, 2017). We study 5 types of positional encodings; *EigVecs-k* corresponds to the smallest non-trivial k eigenvectors, *Rand sign(EigVecs)* randomly flips the sign of the k smallest non-trivial eigenvectors in each batch, *Abs(EigVecs)* takes the absolute value of the k eigenvectors, *Fixed node ordering* uses the original node ordering of graphs, and *Rand node ordering* randomly permutes ordering of nodes in each batch. We observed that the best results are consistently produced with the Laplacian PEs with random sign flipping at training. For index PEs, randomly permuting the ordering of nodes also improves significantly the performances over keeping fixed the original node ordering. However, Laplacian PEs clearly outperform index PEs.

E.2 Edge representations for link prediction.

E.2.1 With GatedGCN and GAT

The TSP and COLLAB edge classification tasks present an interesting empirical result for GCNs: Isotropic models (*vanilla* GCN, GraphSage) are consistently outperformed by their Anisotropic counterparts which use joint representations of adjacent nodes as edge features during aggregation (GAT, GatedGCN). In this section, we systematically study the impact of anisotropy by instantiating three variants of GAT and GatedGCN:
(1) Isotropic aggregation (such as *vanilla* GCNs (Kipf and Welling, 2017)) with node updates of the form:

$$h_i^{\ell+1} = \sigma\big(\sum_{j \in \mathcal{N}_i} W^\ell h_j^\ell \big), \quad \text{identified by } (\textit{E.Feat,E.Repr}{=}\text{✗,✗}) \text{ in Table 14;} \tag{49}$$

(2) Anisotropy using edge features (such as GAT by default (Veličković et al., 2018)) with node updates as:

$$h_i^{\ell+1} = \sigma\big(\sum_{j \in \mathcal{N}_i} f_{V^\ell}(h_i^\ell, h_j^\ell) \cdot W^\ell h_j^\ell \big), \quad \text{with } (\textit{E.Feat,E.Repr}{=}\text{✓,✗}); \tag{50}$$

and (3) Anisotropy with edge features and explicit edge representations updated at each layer with node/edge updates as (such as in GatedGCN by default (Bresson and Laurent, 2017)):

$$h_i^{\ell+1} = \sigma\big(\sum_{j \in \mathcal{N}_i} e_{ij}^\ell \cdot W^\ell h_j^\ell \big),\ e_{ij}^{\ell+1} = f_{V^\ell}\big(h_i^\ell, h_j^\ell, e_{ij}^\ell\big), \quad \text{with } (\textit{E.Feat,E.Repr}{=}\text{✓,✓}). \tag{51}$$

The formal update equations of the three variants of **GatedGCN** are:
Isotropic, similar to vanilla GCNs with sum aggregation:

$$h_i^{\ell+1} = h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell h_i^\ell + \sum_{j \in \mathcal{N}_i} V^\ell h_j^\ell\Big)\Big), \quad \text{where } U^\ell, V^\ell \in \mathbb{R}^{d \times d}. \tag{52}$$

Anisotropic with intermediate edge features computed as joint representations of adjacent node features at each layer:

$$h_i^{\ell+1} = h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell h_i^\ell + \sum_{j \in \mathcal{N}_i} e_{ij}^\ell \odot V^\ell h_j^\ell\Big)\Big), \tag{53}$$

$$e_{ij}^\ell = \frac{\sigma(\hat{e}_{ij}^\ell)}{\sum_{j' \in \mathcal{N}_i} \sigma(\hat{e}_{ij'}^\ell) + \varepsilon}, \quad \hat{e}_{ij}^\ell = A^\ell h_i^{\ell-1} + B^\ell h_j^{\ell-1}, \tag{54}$$

where $U^\ell, V^\ell \in \mathbb{R}^{d \times d}$, $\odot$ is the Hadamard product, and e_{ij}^ℓ are the edge gates.
Anisotropic with edge features as well as explicit edge representations updated across layers in addition to node features, as in GatedGCN by default, Eq.(16):

$$h_i^{\ell+1} = h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell h_i^\ell + \sum_{j \in \mathcal{N}_i} e_{ij}^\ell \odot V^\ell h_j^\ell\Big)\Big), \tag{55}$$

$$e_{ij}^\ell = \frac{\sigma(\hat{e}_{ij}^\ell)}{\sum_{j' \in \mathcal{N}_i} \sigma(\hat{e}_{ij'}^\ell) + \varepsilon}, \tag{56}$$

$$\hat{e}_{ij}^\ell = \hat{e}_{ij}^{\ell-1} + \text{ReLU}\Big(\text{BN}\big(A^\ell h_i^{\ell-1} + B^\ell h_j^{\ell-1} + C^\ell \hat{e}_{ij}^{\ell-1}\big)\Big), \tag{57}$$

where $U^\ell, V^\ell \in \mathbb{R}^{d\times d}$, $\odot$ is the Hadamard product, and e^ℓ_{ij} are the edge gates. The input edge features from the datasets (*e.g.* distances for TSP, collaboration year and frequency for COLLAB) can optionally be used to initialize the edge representations $\hat{e}^{\ell=0}_{ij}$. Note that there may be a multitude of approaches to instantiating anisotropic GNNs and using edge features (Battaglia et al., 2016; Sanchez-Gonzalez et al., 2018; Brockschmidt, 2019) besides the ones we consider.

The formal update equations of the three variants of **GAT** are:
Isotropic, similar to multi-headed vanilla GCNs with sum aggregation:

$$h_i^{\ell+1} = \text{Concat}_{k=1}^K\Big(\text{ELU}\Big(\text{BN}\Big(\sum_{j\in\mathcal{N}_i} U^{k,\ell}\ h_j^\ell\Big)\Big)\Big), \quad \text{where } U^{k,\ell}\in\mathbb{R}^{\frac{d}{K}\times d}. \tag{58}$$

Anisotropic with intermediate edge features computed as joint representations of adjacent node features at each layer, as in GAT by default, Eq.(10):

$$\begin{aligned} h_i^{\ell+1} &= h_i^\ell + \text{ELU}\Big(\text{BN}\Big(\text{Concat}_{k=1}^K\Big(\sum_{j\in\mathcal{N}_i} e_{ij}^{k,\ell}\ U^{k,\ell}\ h_j^\ell\Big)\Big)\Big), && (59)\\ e_{ij}^{k,\ell} &= \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}_{ij'}^{k,\ell})}, \quad \hat{e}_{ij}^{k,\ell} = \text{LeakyReLU}\Big(V^{k,\ell}\ \text{Concat}\big(U^{k,\ell}h_i^\ell,\ U^{k,\ell}h_j^\ell\big)\Big), && (60)\end{aligned}$$

where $U^{k,\ell}\in\mathbb{R}^{\frac{d}{K}\times d}, V^{k,\ell}\in\mathbb{R}^{\frac{2d}{K}}$ are the K linear projection heads and $e_{ij}^{k,\ell}$ are the attention coefficients for each head.
Anisotropic with edge features as well as explicit edge representations updated across layers in addition to node features:

$$\begin{aligned} h_i^{\ell+1} &= h_i^\ell + \text{ELU}\Big(\text{BN}\Big(\text{Concat}_{k=1}^K\Big(\sum_{j\in\mathcal{N}_i} a_{ij}^{k,\ell}\ U^{k,\ell}\ h_j^\ell\Big)\Big)\Big), && (61)\\ e_{ij}^{\ell+1} &= e_{ij}^\ell + \text{ELU}\Big(\text{BN}\Big(\text{Concat}_{k=1}^K\Big(B^{k,\ell}\ \text{Concat}\big(A^{k,\ell}e_{ij}^\ell,\ U^{k,\ell}h_i^\ell,\ U^{k,\ell}h_j^\ell\big)\Big)\Big)\Big), && (62)\\ a_{ij}^{k,\ell} &= \frac{\exp(\hat{a}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{a}_{ij'}^{k,\ell})}, && (63)\\ \hat{a}_{ij}^{k,\ell} &= \text{LeakyReLU}\Big(V^{k,\ell}\ \text{Concat}\big(A^{k,\ell}e_{ij}^\ell,\ U^{k,\ell}h_i^\ell,\ U^{k,\ell}h_j^\ell\big)\Big), && (64)\end{aligned}$$

where $U^{k,\ell}\in\mathbb{R}^{\frac{d}{K}\times d}, V^{k,\ell}\in\mathbb{R}^{\frac{3d}{K}}, A^{k,\ell}\in\mathbb{R}^{\frac{d}{K}\times d}, B^{k,\ell}\in\mathbb{R}^{\frac{d}{K}\times\frac{3d}{K}}$ are the K linear projection heads and $a_{ij}^{k,\ell}$ are the attention coefficients for each head. The input edge features from the datasets can optionally be used to initialize the edge representations $e_{ij}^{\ell=0}$.

Numerical Experiments and Analysis

In Table 14, we show the experiments of the three variants of GatedGCN and GAT on TSP and COLLAB. GatedGCN-E and GAT-E in Table are models using input edge features from the datasets to initialize the edge representations e_{ij}. As maintaining edge representations comes with a time and memory cost for the large COLLAB graph, all models use a reduced budget of 27K parameters to fit the GPU memory, and are allowed to train for a maximum of 24 hours for convergence.

	Model	E.Feat.	E.Repr.	L	#Param	Test Acc.±s.d.	Train Acc.±s.d.	#Epochs	Epoch/Total
TSP	GatedGCN	×	×	4	99026	0.646±0.002	0.648±0.002	197.50	150.83s/8.34hr
		✓	×	4	98174	0.757±0.009	0.760±0.009	218.25	197.80s/12.06hr
		✓	✓	4	97858	0.791±0.003	0.793±0.003	159.00	218.20s/9.72hr
	GatedGCN-E	✓	✓	4	97858	**0.808±0.003**	0.811±0.003	197.00	218.51s/12.04hr
	GAT	×	×	4	95462	0.643±0.001	0.644±0.001	132.75	325.22s/12.10hr
		✓	×	4	96182	0.671±0.002	0.673±0.002	328.25	68.23s/6.25hr
		✓	✓	4	96762	0.748±0.022	0.749±0.022	93.00	462.22s/12.10hr
	GAT-E	✓	✓	4	96762	0.782±0.006	0.783±0.006	98.00	438.37s/12.11hr
COLLAB	GatedGCN	×	×	3	26593	35.989±1.549	60.586±4.251	148.00	263.62s/10.90h
		✓	×	3	26715	50.668±0.291	96.128±0.576	172.00	384.39s/18.44hr
		✓	✓	3	27055	**51.537±1.038**	96.524±1.704	188.67	376.67s/19.85hr
	GatedGCN-E	✓	✓	3	27055	47.212±2.016	85.801±0.984	156.67	377.04s/16.49hr
	GAT	×	×	3	28201	41.141±0.701	70.344±1.837	153.50	371.50s/15.97hr
		✓	×	3	28561	50.662±0.687	96.085±0.499	174.50	403.52s/19.69hr
		✓	✓	3	26676	49.674±0.105	92.665±0.719	201.00	349.19s/19.59hr
	GAT-E	✓	✓	3	26676	44.989±1.395	82.230±4.941	120.67	328.29s/11.10hr

Table 14: Study of anisotropy and edge representations for link prediction on TSP and COLLAB. **Red**: the best model, Violet: good models.

On both TSP and COLLAB, upgrading isotropic models with edge features significantly boosts performance given the same model parameters (*e.g.* 0.75 vs. 0.64 F1 score on TSP, 50.6% vs. 35.9% Hits@50 on COLLAB for GatedGCN with edge features vs. the isotropic variant). Maintaining explicit edge representations across layers further improves F1 score for TSP, especially when initializing the edge representations with euclidean distances between nodes (*e.g.* 0.78 vs. 0.67 F1 score for GAT-E vs. standard GAT). On COLLAB, adding explicit edge representations and inputs degrades performance, suggesting that the features (collaboration frequency and year) are not useful for the link prediction task (*e.g.* 47.2 vs. 51.5 Hits@50 for GatedGCN-E vs. GatedGCN). As suggested by Hu et al. (2020), it would be interesting to treat COLLAB as a multi-graph with temporal edges, motivating the development of task-specific anisotropic edge representations beyond generic attention and gating mechanisms.

E.2.2 With GraphSage

Interestingly, in Table 5 for COLLAB, we found that the isotropic GraphSage with max aggregation performs close to GAT and GatedGCN models, both of which perform anisotropic mean aggregation. On the other hand, models which use sum aggregation (GIN, MoNet) are unable to beat the simple matrix factorization baseline. This result indicates that aggregation functions which are invariant to node degree (max and mean) provide a powerful inductive bias for COLLAB.

We instantiate two anisotropic variants of GraphSage, as described in the following paragraphs, and compare them to GAT and GatedGCN on COLLAB in Table 15. We find that upgrading max aggregators with edge features does not significantly boost performance. On the other hand, maintaining explicit edge representations across layers hurts the models, presumably due to using very small hidden dimensions. (As previously mentioned, maintaining representations for both 235K nodes and 2.3M edges leads to significant GPU memory usage and requires using smaller hidden dimensions.)

Model	Edge Feat.	Edge Repr.	Aggregation Function	L	#Param	Test Acc. ±s.d.	Train Acc. ±s.d.	#Epoch	Epoch/ Total
GatedGCN	×	×	Sum	3	26593	35.989±1.549	60.586±4.251	148.00	263.62s/10.90h
	✓	×	Weighted Mean	3	26715	50.668±0.291	96.128±0.576	172.00	384.39s/18.44hr
	✓	✓	Weighted Mean	3	27055	**51.537±1.038**	96.524±1.704	188.67	376.67s/19.85hr
GatedGCN-E	✓	✓	Weighted Mean	3	27055	47.212±2.016	85.801±0.984	156.67	377.04s/16.49hr
GAT	×	×	Sum	3	28201	41.141±0.701	70.344±1.837	153.50	371.50s/15.97hr
	✓	×	Weighted Mean	3	28561	50.662±0.687	96.085±0.499	174.50	403.52s/19.69hr
	✓	✓	Weighted Mean	3	26676	49.674±0.105	92.665±0.719	201.00	349.19s/19.59hr
GAT-E	✓	✓	Weighted Mean	3	26676	44.989±1.395	82.230±4.941	120.67	328.29s/11.10hr
GraphSage	×	×	Max	3	26293	50.908±1.122	98.617±1.763	157.75	241.49s/10.62hr
	✓	×	Weighted Max	3	26487	50.997±0.875	99.158±0.694	112.00	366.24s/11.46hr
	✓	✓	Weighted Max	3	26950	48.530±1.919	90.990±9.273	118.25	359.18s/11.88hr
GraphSage-E	✓	✓	Weighted Max	3	26950	47.315±1.939	93.475±5.884	120.00	359.10s/12.07hr

Table 15: Study of anisotropic edge features and representations for link prediction on COLLAB, including GraphSage models. **Red**: the best model, Violet: good models.

Isotropic, as in GraphSage by default, Eq.(8):

$$h_i^{\ell+1} = h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell \text{ Concat }\Big(h_i^\ell,\ \text{Max}_{j\in\mathcal{N}_i} \text{ ReLU }\Big(V^\ell h_j^\ell\Big)\Big)\Big)\Big), \tag{65}$$

where $U^\ell \in \mathbb{R}^{d\times 2d}, V^\ell \in \mathbb{R}^{d\times d}$.

Anisotropic with intermediate edge features computed as joint representations of adjacent node features at each layer:

$$\begin{aligned} h_i^{\ell+1} &= h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell \text{ Concat }\Big(h_i^\ell,\ \text{Max}_{j\in\mathcal{N}_i} \text{ ReLU }\Big(\sigma\left(e_{ij}^\ell\right)\odot V^\ell h_j^\ell\Big)\Big)\Big)\Big), & (66)\\ e_{ij}^\ell &= A^\ell\left(h_i^{\ell-1}+h_j^{\ell-1}\right), & (67)\end{aligned}$$

where $U^\ell \in \mathbb{R}^{d\times 2d}, V^\ell, A^\ell \in \mathbb{R}^{d\times d}$, $\odot$ is the Hadamard product, and e_{ij}^ℓ are the edge gates.

Anisotropic with edge features as well as explicit edge representations updated across layers in addition to node features:

$$\begin{aligned} h_i^{\ell+1} &= h_i^\ell + \text{ReLU}\Big(\text{BN}\Big(U^\ell \text{ Concat }\Big(h_i^\ell,\ \text{Max}_{j\in\mathcal{N}_i} \text{ ReLU }\Big(\sigma\left(\hat{e}_{ij}^\ell\right)\odot V^\ell h_j^\ell\Big)\Big)\Big)\Big), & (68)\\ \hat{e}_{ij}^\ell &= A^\ell\left(h_i^{\ell-1}+h_j^{\ell-1}\right) + B^\ell e_{ij}^{\ell-1}, \quad e_{ij}^{\ell+1} = e_{ij}^\ell + \text{ReLU}\Big(\text{BN}\Big(\hat{e}_{ij}^\ell\Big)\Big), & (69)\end{aligned}$$

where $U^\ell \in \mathbb{R}^{d\times 2d}, V^\ell, A^\ell, B^\ell \in \mathbb{R}^{d\times d}$, $\odot$ is the Hadamard product, and $\hat{e}_{ij}^\ell$ are the edge gates. The input edge features from the datasets can optionally be used to initialize the edge representations $e_{ij}^{\ell=0}$.

Appendix F. Experiments on TU datasets

Apart from the proposed datasets in our benchmark (Section C), we perform experiments on 3 TU datasets for graph classification – ENZYMES, DD and PROTEINS. Our goal is to empirically highlight some of the challenges of using these conventional datasets for benchmarking GNNs.

	Model	L	#Param	seed 1				seed 2			
				Test Acc.±s.d.	Train Acc.±s.d.	#Epoch	Epoch/Total	Test Acc.±s.d.	Train Acc.±s.d.	#Epoch	Epoch/Total
ENZYMES	MLP	4	101481	55.833±3.516	93.062±7.551	332.30	0.18s/0.17hr	53.833±4.717	87.854±10.765	327.80	0.19s/0.18hr
	vanilla GCN	4	103407	**65.833±4.610**	97.688±3.064	343.00	0.69s/0.67hr	64.833±7.089	93.042±4.982	334.30	0.74s/0.70hr
	GraphSage	4	105595	65.000±4.944	100.000±0.000	294.20	1.62s/1.34hr	**68.167±5.449**	100.000±0.000	287.30	1.76s/1.42hr
	MoNet	4	105307	63.000±8.090	95.229±5.864	333.70	0.53s/0.49hr	62.167±4.833	93.562±5.897	324.40	0.68s/0.62hr
	GAT	4	101274	**68.500±5.241**	100.000±0.000	299.30	0.70s/0.59hr	**68.500±4.622**	100.000±0.000	309.10	0.76s/0.66hr
	GatedGCN	4	103409	**65.667±4.899**	99.979±0.062	316.80	2.31s/2.05hr	**70.000±4.944**	99.979±0.062	313.20	2.63s/2.30hr
	GIN	4	104864	65.333±6.823	100.000±0.000	402.10	0.53s/0.61hr	67.667±5.831	100.000±0.000	404.90	0.60s/0.68hr
	RingGNN	2	103538	18.667±1.795	20.104±2.166	337.30	7.12s/6.71hr	45.333±4.522	56.792±6.081	497.50	8.05s/11.16hr
	3WLGNN	3	104658	61.000±6.799	98.875±1.571	381.80	9.22s/9.83hr	57.667±9.522	96.729±5.525	336.50	11.80s/11.09hr
DD	MLP	4	100447	72.239±3.854	73.816±1.015	371.80	6.36s/6.61hr	**72.408±3.449**	73.880±0.623	349.60	1.13s/1.11hr
	vanilla GCN	4	102293	72.758±4.083	100.000±0.000	266.70	3.56s/2.66hr	**73.168±5.000**	100.000±0.000	270.20	3.81s/2.88hr
	GraphSage	4	102577	**73.433±3.429**	100.000±0.000	267.20	11.50s/8.59hr	71.900±3.647	100.000±0.000	265.50	6.60s/4.90hr
	MoNet	4	102305	71.736±3.365	81.003±2.593	252.60	3.30s/2.34hr	71.479±2.167	81.268±2.295	253.50	2.83s/2.01hr
	GAT	4	100132	**75.900±3.824**	95.851±2.575	201.30	6.31s/3.56hr	**74.198±3.076**	96.964±1.544	220.10	2.84s/1.75hr
	GatedGCN	4	104165	**72.918±2.090**	82.796±2.242	300.70	12.05s/10.13hr	71.983±3.644	83.243±3.716	323.60	8.78s/7.93hr
	GIN	4	103046	71.910±3.873	99.851±0.136	275.70	5.28s/4.08hr	70.883±2.702	99.883±0.088	276.90	2.31s/1.79hr
	RingGNN	2	109857	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	3WLGNN	3	104124	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
PROTEINS	MLP	4	100643	75.644±2.681	79.847±1.551	244.20	0.42s/0.29hr	75.823±2.915	79.442±1.443	241.20	0.35s/0.24hr
	vanilla GCN	4	104865	76.098±2.406	81.387±2.451	350.90	1.55s/1.53hr	**75.912±3.064**	82.140±2.706	349.60	1.46s/1.42hr
	GraphSage	4	101928	75.289±2.419	85.827±0.839	245.40	3.36s/2.30hr	75.559±1.907	85.118±1.171	244.40	3.44s/2.35hr
	MoNet	4	103858	**76.452±2.898**	78.206±0.548	306.80	1.23s/1.06hr	**76.453±2.892**	78.273±0.695	289.50	1.26s/1.03hr
	GAT	4	102710	**76.277±2.410**	83.186±2.000	344.60	1.47s/1.42hr	75.557±3.443	84.253±2.348	335.10	1.51s/1.41hr
	GatedGCN	4	104855	**76.363±2.904**	79.431±0.695	293.80	5.03s/4.13hr	**76.721±3.106**	78.689±0.692	272.80	4.78s/3.64hr
	GIN	4	103854	74.117±3.357	75.351±1.267	420.90	1.02s/1.20hr	71.241±4.921	71.373±2.835	362.00	1.04s/1.06hr
	RingGNN	2	109036	67.564±7.551	67.607±4.401	150.40	28.61s/12.08hr	56.063±6.301	59.289±5.560	222.70	19.08s/11.88hr
	3WLGNN	3	105366	61.712±4.859	62.427±4.548	211.40	12.82s/7.58hr	64.682±5.877	65.034±5.253	200.40	13.05s/7.32hr

Table 16: Performance on the TU datasets with 10-fold cross validation (higher is better). Two runs of all the experiments using the same hyperparameters but different random seeds are shown separately to note the differences in ranking and variation for reproducibility. The top 3 performance scores are highlighted as **First**, **Second**, **Third**.

Splitting. Since the 3 TU datasets that we use do not have standard splits, we perform a 10-fold cross validation split which gives 10 sets of train, validation and test data indices in the ratio 8 : 1 : 1. We use stratified sampling to ensure that the class distribution remains the same across splits. The indices are saved and used across all experiments for fair comparisons. There are 480 train/60 validation/60 test graphs for ENZYMES, 941 train/118 validation/119 test graphs for DD, and 889 train/112 validation/112 test graphs for PROTEINS datasets in each of the folds.

Training. We use Adam optimizer with a similar learning rate strategy as used in our benchmark's experimental protocol. An initial learning rate is tuned from a range of 1×10^{-3} to 7×10^{-5} using grid search for every GNN models. The learning rate reduce factor is 0.5, the patience value is 25 and the stopping learning rate is 1×10^{-6}.

Performance Measure. We use classification accuracy between the predicted labels and groundtruth labels as our performance measure. The model performance is evaluated on the test split of the 10 folds for all TU datasets, and reported as the average and the standard deviation of the 10 scores.

Our numerical results on the TU datasets – ENZYMES, DD and PROTEINS are presented in Table 16. We observe all NNs have similar statistical test performance as the standard deviation is quite large. We also report a second run of these experiments with the same experimental protocol, *i.e.* the same 10-fold splitting and hyperparameters but different initialization (seed). We observe a change of model ranking, which we attribute to the small size of the datasets and the non-determinism of gradient descent optimizers. We also observe that, for DD and PROTEINS, the graph-agnostic MLP baselines perform as good as

GNNs. Our observations reiterate how experiments on the small TU datasets are difficult to determine which GNNs are powerful and robust.

Appendix G. A Note on Graph Size Normalization

Intuitively, batching graphs of variable sizes may lead to node representation at different scales, making it difficult to learn the optimal statistics μ and σ for BatchNorm across irregular batch sizes and variable graphs. A preliminary version of this work introduced a graph size normalization technique called GraphNorm, which normalizes the node features h_i^ℓ w.r.t. the graph size, *i.e.*,

$$\bar{h}_i^\ell = h_i^\ell \times \frac{1}{\sqrt{\mathcal{V}}}, \tag{70}$$

where $\mathcal{V}$ is the number of graph nodes. The GraphNorm layer is placed before the BatchNorm layer.

We would like to note that GraphNorm does not have any concrete theoretical basis as of now, and was proposed based on initially promising empirical results on datasets such as ZINC and CLUSTER. Future work shall investigate more principled approaches towards designing normalization layers for graph structured data.

Appendix H. Elaboration on Benchmarking Design Choices

In Section 2, we provided a brief overview on the design choices that we had to make to build the proposed benchmarking framework. In particular, the decisions on the selection of the specific graph datasets that we have included in this framework, the necessity to constraint model parameters for comparison of GNNs' performance, and whether a standard codebase with data, training, evaluation pipelines is required can be derived from several reasonings. In this section, we provide an elaborate discussion on these factors and how possible extensions can be developed in future with ease and as per required by a research agenda.

Datasets. Our collection of datasets is based on medium-scale size and criteria of diversity in terms of the end-application domains, learning tasks at graph-, edge-, or node-levels, and their source of construction being real or mathematical. The medium-scale size of datasets enables quick prototyping of novel ideas and robust analysis could be generated in single experiments in as less as 12 hours of maximum time per experiment. Similarly, the diversity ensures a model can be tested on not just one end-application domain but a number of such domains. However, despite the best efforts, after any collection of datasets in such a research area where a general GNN architecture is expected to be robust to a variety of tasks and domains, there could always be need of additional datasets. Due to this necessity, the proposed framework can be extended with new datasets conveniently by any researchers adopting it. We have also observed the open-sourced GitHub repo of our framework being used accordingly with an example repo being `https://github.com/karl-zhao/benchmarking-gnns-pyg` (Zhao et al., 2020) which extends the framework with additional node classification datasets as well as adopts it in Pytorch Geometric (Fey and Lenssen, 2019) instead of DGL. Such adoption of our framework demonstrates its flexibility and the supported convenient extensions. We provide detailed instructions on adding new datasets to the framework in our GitHub repository's `README` .

Parameter Budgets. As we have already mentioned, we designed the framework with the objective that it is used to conveniently 'identify first principles' in GNNs' research and not drive a model towards achieving SOTA performance. To enforce this, a straightforward and sound choice is to constraint model parameters and fix it to a specific number (as eg. 100k and 500k) when comparing two or more GNNs. With this choice, we can likely rely on the inference that performance gains are coming from architectural designs and not merely large trainable parameters. The parameter budgeting also tells that the proposed framework may not be ideal to optimize a model to achieve SOTA by tuning hyperparameters, increasing model size to as much parameters as a server can fit, etc. However, we believe we condition the framework to be suitable for identifying performance trends and infer which first principles work robustly across different model experiments. Once such principles are identified, models can further be scaled without any constraints to achieve SOTA performance targeted benchmarks, beyond the datasets we included here.

Codebase. A major contribution of this work is the release of the open-source coding infrastructure on GitHub. As observed since the first release in March 2020, the framework has been used extensively to develop new ideas in the field. In the existing literature prior to this work (Errica et al., 2019), it was a major issue that different research papers in this field adopted inconsistent model comparison methods. Our framework addresses this need of having a standard codebase that helps in training and evaluating GNNs on a collection of appropriate datasets with consistent settings. While a limiting perspective to such codebase can be that it restricts on the diverse choices which researchers often adopt in deep learning to fully realise the capabilities of a model, we understand that we have set out specific objectives of the need of the proposed coding infrastructure and any extensions with other training settings to the codebase can be done by augmenting methods or modules that applies to each model in a fair and consistent way.

Appendix I. Hardware

Timing research code can be tricky due to differences of implementations and hardware acceleration. Nonetheless, we take a practical view and report the average wall clock time per epoch and the total training time for each model. All experiments were implemented in DGL/PyTorch. We run experiments for MNIST, CIFAR10, ZINC, AQSOL, TSP, COLLAB, WikiCS, CSL, CYCLES, GraphTheoryProp and TUs on an Intel Xeon CPU E5-2690 v4 server with 4 Nvidia 1080Ti GPUs (11 GB), and for PATTERN and CLUSTER on an Intel Xeon Gold 6132 CPU with 4 Nvidia 2080Ti (11 GB) GPUs. Each experiment was run on a single GPU and 4 experiments were run on the server at any given time (on different GPUs). We run each experiment for a maximum of 12 hours.

Appendix J. Memory Usage

For datasets that contain graphs with variable sizes, the memory consumed during training by the GPU device changes at each batch of graphs. We report in Figure 16 the GPU memory consumption during the training of GCN, GAT and GatedGCN on two datasets–ZINC and

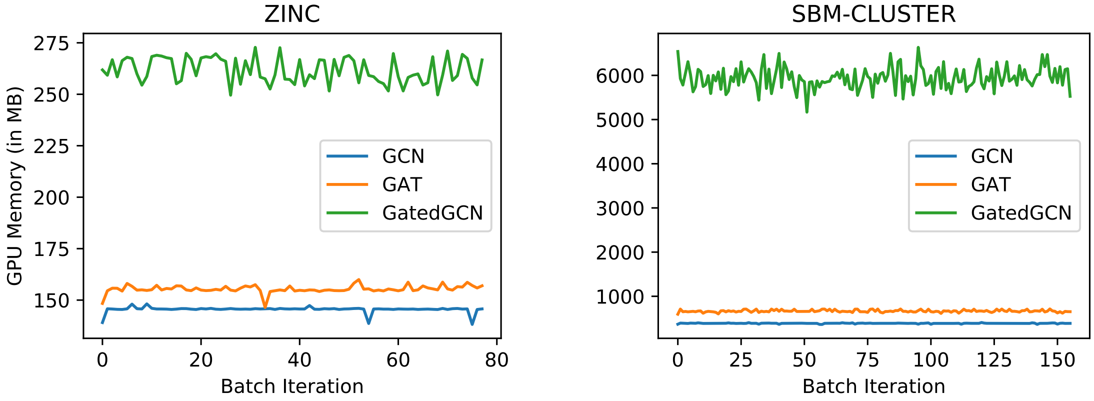


Figure 16: Memory consumed by the GPU device in a forward pass of a batch during training. All GNNs shown here have 500k learnable parameters. Batch size is 128 for ZINC and 64 for SBM-CLUSTER.

SBM-CLUSTER. The plots show the memory allocated during the model's forward pass using a batch of graphs (128 graphs for ZINC and 64 for SBM-CLUSTER) and is computed by using PyTorch's `torch.cuda.memory_allocated(device)` functionality. Overall, it can be observed that GatedGCN is a relatively higher memory-intensive model as compared with GCN and GAT, see Section B.1 for the respective models' equations.