# Review of Benchmarking Graph Neural Networks

Melvin SEVI
Master MVA ENS-Paris Saclay
melvin.sevi@ens-paris-saclay.fr

Akedjou Achraff ADJILEYE
Master MVA ENS-Paris Saclay
akedjou-achraff-brad.adjileye@universite-paris-saclay.fr

## ABSTRACT

Graph Neural Networks (GNNs) have emerged as formidable tools for analyzing and learning from graph structured data. They have found applications in diverse fields including social networks analysis, bio-informatics, recommendation systems, image recognition, natural language processing or 3D shape analysis. The versatility of graphs as a representation framework enables them to capture the underlying structures of a wide range of complex problems. However, with the ever-expanding amount of GNN models designed for graph-based learning, it becomes imperative to assess their performance and compare them systematically for informed model selection and development. This is precisely the focus of Dwivedi V P. et al. in [1]. In this report, we go into a comprehensive analysis of their work, trying to understand its contributions and limitations. Note that [1] only focus on undirected graphs.

**ACM Reference Format:**
Melvin SEVI and Akedjou Achraff ADJILEYE. 2023. Review of Benchmarking Graph Neural Networks. In *Proceedings of MVA 2023.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 BENCHMARKING CONTEXT

(1) **Why should we benchmark machine learning (ML) models?** With the ever-expanding amount of neural network model architectures designed for each machine learning task on specific type of data (image, text, graph, etc), it becomes imperative to assess their performance and compare them systematically for informed model selection and development.
(2) **Which family of models to benchmark?** Since two models can handle the same data and do the same task, they are comparable in terms of performance on a metric of interest. For instance, two molecular graphs classification models can be compared in terms of accuracy.
(3) **Computational resources**: To make a good and fair comparison, it's important to have a way to compare models computational resources. Since we're interested in neural networks based model, a natural criterion can be the models size in terms of number of parameters. All the compared models should have their sizes in the same range to ensure fairness.
(4) **Parameters repartition**: Assume that we have two neural networks with the same number of parameters. Is that enough to say that we can fairly compare the two model? An important thing to take into account is the models hidden structure. How to ensure that the models to compare have same depth (number of layers) and same width (number of neurons per layer) ?
(5) **Training parameters**: To make fairly comparison and consequently have a good benchmark, a natural idea is to think that models should be trained with the same parameters calibration, for the same time and with the same stopping criterion. But ensuring this point can be complicated because the models don't have the same architecture and therefore not the same objective function landscape. Thus, one model may perform better than another not because it is better itself, but because the training parameters used for benchmarking are better suited to it. This is a very important point that could be a big limitation for empirical benchmarks.
(6) **Exhaustiveness of the benchmarking datasets**: Saying a machine learning model $A$ is better than another one $B$ requires to specify the evaluation dataset. However, it's common to refer to a task instead of a dataset and say for example that "a model X is the state of the art (SOTA) for Y task". This make sense if the dataset is sufficiently representative of the task. Hence, it's important to have exhaustive datasets to make a good benchmark.
(7) **Benchmarking Framework** Instead of just comparing models and report the performance metrics, a good benchmark implies to develop framework with well documented and reproductible code. That given, it should be easy to reproduce experiment on a new model and add it to the leaderboard, or to evaluate the impact of a change on models performance.

## 2 GRAPH NEURAL NETWORK (GNNS) MODELS

As in response in (2) of the precedent section, the authors of [1] aimed to benchmark the family of graph neural networks (GNNs). A GNN can be divided in the three parts: the input layer, the hidden layers and the output layer.

Authors explore two family of GNNs that differ by the way the graph are feed to their input layers: Message-Passing Graph Convolutional Networks (**MP-GCNs**) and Weisfeiler-Lehman Graph Neural Networks (**WL-GNNs**).

(1) A MP-GCN 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 $\alpha_i$ and $\beta_{ij}$ are embedded into $d$-dimensional hidden features $h_i^0$ and $e_{ij}^0$ via a simple linear


*MVA 2023, December 2023, Paris, FR*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

projection before passing them to a graph neural network:

$$h_i^0 = U^0 \alpha_i + u^0, \quad e_{ij}^0 = V^0 \beta_{ij} + v^0.$$

where $U^0 \in \mathbb{R}^{d \times a}$, $V^0 \in \mathbb{R}^{de \times b}$, 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. So now nodes have $d$ features and edges have $d_e$ features .

Six (06) MP-GCNs was explored in the benchmark depending on their hidden layers architectures: Graph ConvNet (**GCN**), **GraphSage**, Graph Attention Network (**GAT**), **MoNet**, Gated Graph ConvNet (**GatedGCN**) and Graph Isomorphism Networks (**GIN**). They involve aggregating features from neighboring nodes and edges according to the general formula:

$$h_i^{(l+1)} = f\left(h_i^{(l)}, \{h_j^{(l)}\}_{j \in \mathcal{N}_i}\right)$$

where $\{j \rightarrow i\}$ denotes the set of neighboring nodes $j$ pointed to node $i$, which can be replaced by $\{j \in N_i\}$, the set of neighbors of node $i$. The formula of $f$ for each MP-GCN can be found on [1] (pages 8-13).
Those local aggregations helps nodes in the graph understand their local structure and allows information to flow across the graph. Information sharing allows to discover hidden patterns, which is crucial for various tasks on graphs.

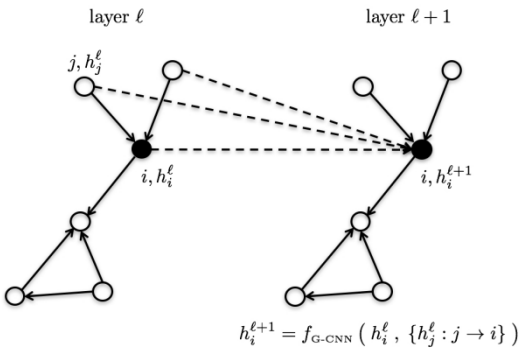


**Figure 1: Illustration of a graph convolutional network layer**

(2) Also, two (02) WL-GNN was explored depending on their hidden layers architectures: **3WL-GNNs** [10] and **RingGNNs**. The main goal of those architectures was to make them as expressive as the famous k-Wesfeiler Leiman test wich was introduced in . The Weisfeiler-Lehman algorithm is a graph isomorphism test that iteratively refines and labels the vertices of graphs. It is commonly used in computational chemistry and computer science for solving problems related to graph structures and isomorphism detection and deeply inspired GNN's archit. Those 3WL-GNN were made because the classical GNN's were not able to distinct isoomorphic graphs. All detailed about the hidden layers formula of 3WL-GNN's can be found in [1] (pages 14-15).

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 $h_{\text{edge}} \in \mathbb{R}^{n \times n \times de}$, the input tensor to the RingGNN and 3WL-GNN networks is defined as

$$h^0 \in \mathbb{R}^{n \times n \times (1+d+de)}$$

where

$$h_{i,j,1}^0 = A_{ij} \in \mathbb{R}, \quad \forall i, j$$

$$h_{i,j,2:d+1}^0 = \begin{cases} h_{\text{node},i} \in \mathbb{R}^d, & \text{for } i = j, \\ \mathbf{0}, & \text{for } i \neq j, \end{cases} \quad \forall i, j$$

$$h_{i,j,d+2:d+de+1}^0 = h_{\text{edge},ij} \in \mathbb{R}^{de}, \quad \forall i, j.$$

For these two families of graph neural networks, the output layer of don't depend on the model itself, but on the task it's aimed to perform. Thus, for any of the eight (08) GNN model, the output layer will always be one of the following four: graph classifier layer, graph regression layer, node classifier layer, node regression layer. The input of the output layer is the result of the final MP-GCN/WL-GNN hidden layer for each node of the graph (except GIN, which uses features from all intermediate layers). This is used to feed to the final multi-layers perceptron (MLP) layers to perform the desired task. Detailed formula about the four output layers type can be found in [1] (pages 13, 15-16).

# 3 DATASETS, MODELS TRAINING, FAIRNESS TEST

## 3.1 Used fairness training procedure

- Adam optimizer with the same learning rate decay strategy was used 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.
- The training is stopped either when the learning rate has reached the small value of $10^{-6}$ (or $10^{-5}$ for in certain cases), or the computational time reaches 12 hours.
- Each experiment was runned with 4 different seeds and the statistics of the 4 results are reported.

## 3.2 Computational budget

As the goal to compare and benchmark the model and their building blocks within a budget of parameters, the authors uses two parameter budgets: - 100k parameters for each GNN for all the tasks - 500k parameters for GNNs for which they investigated scaling to larger parameters and deeper layers.

It's important to note that the number of hidden layers and hidden dimensions are selected accordingly to match these budgets.

## 3.3 Datasets exhaustiveness analysis

Twelve (12) dataset were used in this benchmark, covering the following tasks: 2 datasets on graphs regression, 2 on link prediction (or edge classification), 3 on nodes classifcation, 1 on multi-task graph properties prediction, 1 on cycle detection in graphs and 3

on graphs classification.

The GNNs mentioned in Section 2 were tested on the twelve datasets, and the results are presented with details in [1] (pages 16-29). Each dataset is described clearly, including the dataset's relevance for the benchmark, the performed task, the training and evaluation methods employed, and the obtained results. Our focus in this work is not to report these results and analyze each of them, but to highlight what we found interesting (positive aspects as negative ones) in general and according to our interest.

**Dataset exhaustiveness**: Are the proposed datasets are good and general enough to make a benchmark on graph neural network ? In [1] (section D), the authors claim that "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". It seems like they are justifying that proposition by the fact that a simple graph agnostic multi-layer perceptron (MLP) which updates each node of a given graph independent of one other, $h_i^{(l+1)} = \sigma(W^{(l)} h_i^{(l)})$ presents consistently low scores across all datasets. But in five of the 9 datasets on which they compare a simple MLP to the GNNs, MLP outperforms or performs on par with at least one GNN architecture.

For instance, on the PATTERN datasets ([1], section C.6 tab 8) for node classification, a simple MLP 100k performs on par with a GraphSage 100K and more significant, with a GraphSage 500k, by being +9x and +38x more fast respectively, in terms of epoch duration.

Better, on the MNIST dataset for graph classification, a simple MLP 100k outperforms GCN 100k and MoNet 100k with equal size by being +1.5x and +3x more fast respectively, and more significant, it also outperforms a RingGNN 500k. On CIFAR10, a simple MLP outperforms several architecture by being very more fast, including GCN, MoNet, GIN and RingGNN.

Additionally, the better accuracies on MNIST and CIFAR10 were achieved by GatedGCN with 97.340 and 67.312 respectively and by comparison to the state-of-the-art on these datasets, the better ever accuracy achieved on MNIST is 99.87 [2] and on CIFAR10 is 99.9 [3]. The considered GNNs in this benchmark are far from the SOTA and it's normal to ask what is the real interest of these datasets in a benchmark on GNNs?

## 3.4 Training fairness analysis

For fairness in model comparison, the benchmark authors ensured that the compared models have the same number of parameters, were trained with the same settings (training parameters and stopping criteria). However, is this really a way to ensure fairness? For instance, even when training all models with the same learning rate, how can we guarantee that the chosen value of the learning rate is not inherently more suitable for one model than another? This raises concerns about avoiding favoritism in a specific model convergence due to the choice of training parameters. Given the fact that there is no prior information of each GNN's objective function landscape, such guarantees seem impossible. While using the same training method initially appears as a good way to ensure fairness, upon reflection, we believe this may not necessarily be the case. Imagine we modify the default values of training parameters and keep them constant across all architectures, would the benchmark remain the same? Or, if we increased the training time to allow networks to converge further, would the benchmark order be disrupted? **(a)**

Furthermore, the authors compared various GNN architectures with a fixed budget of 100k parameters. In their experiments, the size (number of layers) and depth of the networks were chosen to fit within this parameter budget. However, a question remains: should one create shallow but wide networks or deep but narrow networks? In most of their experiments, the authors focused more on the number of layers in the networks, consistently ensuring that all networks of the same size (typically 100k parameters) had the same depth. But does this choice have an impact on the benchmark? By altering the depth/width structure of GNNs while keeping the number of parameters fixed at 100k, would the benchmark order still hold? **(b)**

All these questions pose a broader issue regarding the best way to conduct an empirical benchmark. While it seems normal to compare models with the same number of parameters, should we restrict ourselves to training these models with the same parameters for the same duration with the same depth/width structure? Or should we simply train each model optimally and compare performance in terms of effectiveness and efficiency? We think this question needs a careful consideration.

## 3.5 Experimentation using the benchmarking framework

The authors provide clean, user-friendly code that allows for easy reproduction of their results. Therefore, in order to address some of the concerns mentioned earlier regarding fairness, we conducted tests on the **AQSOL** dataset from the benchmark.

**About AQSOL** It's based on AqSolDB [2] 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 LogS units in AqSolDB. These final values are used as the property to regress in the AQSOL dataset. 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. There is no specific motivation for the choice of this dataset other than its small size, which is suitable for the limited computational capacity at our disposal (1 GPU T4 with limited access time, free version of Google Colab). We used the same split (training-validation-test) and the same training method as the benchmark authors ([1], page 19).

**Architecture changes:** As mentioned in Section 3.4 **(b)**, our goal is to assess the robustness of the benchmark to the choice of GNN architecture (depth&width), in the range of the 100k fixed size. In their experiments, the authors set the number of hidden layers to $L = 4$ and chose the width of the layers (number of neurons) accordingly. In our test, we fixed the depth of all networks to $L = 8$

**Table 1: Benchmarking results for AQSOL (lower better), for GAT, the hidden layer (H) structure is different; e.g. 4818 is for 4 layers, 8 attention heads and 18 embedding dimension, the rank of each model is displayed in brackets beside his test MAE**

| Model | Params (4L) | H (4L) | Train MAE (4L)±std | Test MAE(4L)±std | Params (8L) | H (8L) | Train MAE (8L) | Test MAE (8L) |
|---|---|---|---|---|---|---|---|---|
| GCN | 108 442 | 145 | 0.593±0.030 | 1.372±0.020(2) | 108 270 | 107 | 0.864 | 2.151(6) |
| GraphSage | 109 620 | 108 | 0.666±0.027 | 1.431±0.010(4) | 105 442 | 77 | 0.56 | 1.724 (2) |
| MoNet | 109 332 | 90 | 0.557±0.022 | 1.395±0.027(3) | 106 769 | 64 | 0.375 | **1.606**(1) |
| GIN | 107 149 | 110 | 0.660±0.027 | 1.894±0.024(6) | 104 275 | 77 | 0.564 | 1.862(5) |
| GAT | 108 289 | 4818 | 0.678±0.021 | 1.441±0.023(5) | 104 313 | 8426 | 0.658 | 1.842(4) |
| GatedGCN | 108 325 | 70 | 0.576±0.056 | **1.352±0.034**(1) | 108 550 | 50 | 0.764 | 1.808 (3) |

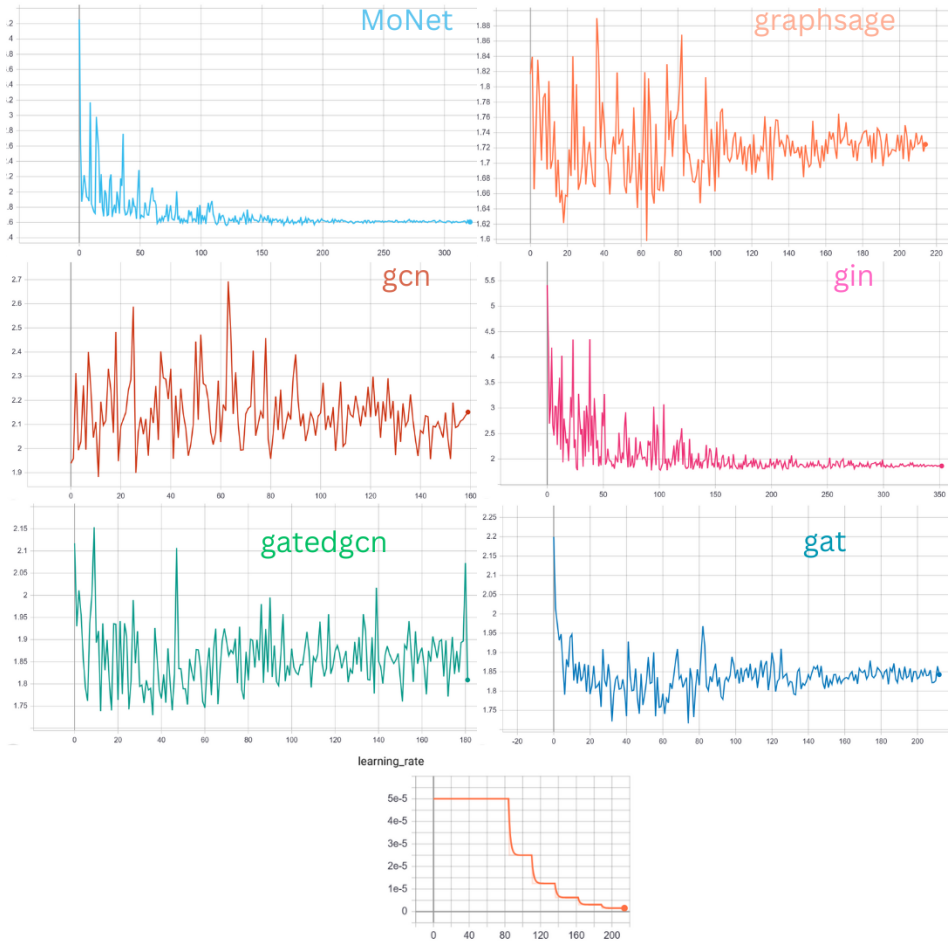


**Figure 2: Test mean absolute error (MAE) curves, the learning rate scheduler is the same for all models**

and chose the number of neurons to maintain the experiment's parameter count (100k). We followed the same training method as the authors (described in Section 3.1). The obtained results are documented in Table 1. Due to limited resources, we tested only the 6 GCN architectures presented in Section 2, and we ran the experiments only once, as opposed to 4 times as in [1].

**Results Analysis:** Table 1 show the lack of robustness of the benchmark to the choice of network structure. The best Mean Absolute Error (MAE) was achieved in [1] by the GatedGCN model, followed by GCN and MoNet to complete the top 3 with comparable scores (a 0.043 difference between the top 3). However, by altering the structure of the hidden layers, MoNet climbs 2 positions to the top, followed by GraphSage, which also gains 2 positions, and GatedGCN, which loses 2 positions. More significantly, the MAE gap between the top 3 increases to 0.202, which is +4.5x more than

before. Furthermore, GCN, which was second with 4 layers, ends up last when increased to 8 layers, despite keeping the number of parameters fixed at 100k. Moreover, the MAE degrades significantly, reaching the worst value obtained in all experiments, 2.156.

Furthermore, the analysis of the test MAE curves (Fig. 2) during models training helps address our concerns about using the same training parameters, especially the same learning rate scheduler, for all models. It can be observed that only the curves of MoNet and GIN look like the desired learning curves. This raises questions about the calibration of the learning rate during model training, in a benchmarking context. For instance, even though GraphSage performs well in our tests, upon observing the test MAE curve, it suggests that better tuning of the training speed could reduce the MAE's variance during training and thus improve model convergence and performance.

# 4 BEYOND BENCHMARKING: USAGE OF THE BENCHMARKING FRAMEWORK TO MAKE IMPROVEMENT

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. In the following section, we'll present some of new explorations made by the authors to improve the GNNs performance on some datasets; highlighting their impactful results.

**The power of Laplacian positional encoding:** In their thorough benchmarking analysis, the researchers observed that the GatedGCN architecture consistently outperforms other architectures across various tasks and datasets. Despite this success, their objective was to enhance these performances by addressing a critical issue with MP-GCN's. The challenge lies in the fact that while message passing enables nodes in a graph to be aware of their local surroundings and structure, it lacks consideration for the positional information of nodes within the graph. For example, if two nodes share identical local structures, a GNN might struggle to differentiate between them, making it less effective for tasks such as detecting graph isomorphism or performing graph classification/regression. The authors propose to compare the performance by incorporating nodes positional encoding in the graph. Unfortunately there is still no theory that guides the choice of positional encodings depending of the class of graphs and tasks. We can employ node indexing, but this is inherently dependent on permutations. The objective is to establish more universally applicable positional encodings that are resilient across diverse graphs and tasks.

## 4.1 Lalacian positional encoding

A widely recognized approach involves using the eigenvectors of the graph Laplacian. This technique draws inspiration from graph spectral theory and is highly related to graph fourier transform.

The Laplacian of an undirected graph $G = (V, E)$, where $V$ is the set of nodes and $E$ is the set of edges, is defined as $L = D - A$. Here, $D$ is the degree matrix of the graph, and $A$ is it's adjacency matrix.The Laplacian matrix is integral to understanding the flow within a graph, providing a measure of smoothness and connectivity. Consider a function $h : V \longrightarrow \mathbb{R}$ on the graph $G$ and L the Laplacian of this graph. The coefficient $i$ of $Lh$ wich is

$$(Lh)_i = h_i - \sum_j A_{ij} * h_j = \sum_j A_{ij} * (h_i - h_j)$$

reflects the influence of the value of the node $h_i$ on the graph. If we sum this value for all the nodes we get the quantity:

$$E(h) = \frac{1}{2} \sum_{(i,j) \in E} A_{ij} (h_i - h_j)^2$$

This function is also commonly referred to as the Dirichlet energy of the graph. When the nodes propagates their value uniformly across the graph, this value tends to zero, intuitively illustrating the Laplacian connection to the smoothness of the graph. A normalized version of the Laplacian matrix is generally used in practice to avoid computations from blowing up after successive neural network layers. The normalized Laplacian, which is equal to

$$D^{-1/2}(D - A)D^{-1/2} = I - D^{-1/2}AD^{-1/2}$$

.

## 4.2 Laplacian Eigenvectors as Graph Fourier Basis

The Laplacian matrix of a graph $G$ is a symmetric and semi-definite matrix (Can be verified by calculating $f^T Lf$ for all vectors $f$), making it diagonalizable in an orthonormal basis with $n$ real positive eigenvalues (where $n$ is the number of nodes in the graph). The eigenvectors corresponding to these eigenvalues are commonly referred to as "the Fourier basis functions" of the graph. It can be demonstrated that any signal on the graph can be effectively decomposed using the eigenvectors of the graph Laplacian since this set of eigenvectors forms a orthonormal basis for $\mathbb{R}^n$. Signals on the graph can be projected onto these eigenvectors, with each eigenvector representing a unique frequency (see figure 5). These eigenvectors of dimension n assign a value to each node of the graph.

When you plot the first few eigenvectors of a line graph, you get the cosine and sine functions, representing the Fourier basis functions with different frequencies. This surprisingly explains the link with the sinusoidal positional encoding used in Transformers architecture as a sentence can be considered as a line graph, word tokens being the nodes and a word being linked to the previous and next word of the sentence.

The first value of each eigenvector are linked to the "first" node, the second one to the second node, and so on. This helps describe the signals value of each node for different frequencies, suggesting that these rows carry a lot of information about the nodes.

We can establish a profound connection between eigenvectors and graph convolutions, drawing a notable **analogy** with the Fourier transform theorem from functional analysis. We believe that this part is crucial to understand the link between laplacian eigenvectors and GCN's. Let $\phi$ be a matrix containing the eigenvectors of the normalized Laplacian and $\Lambda$ a diagonal matrix containing the corresponding eigenvalues, such that $L = \phi \Lambda \phi^T$, where $L$ is the Laplacian matrix.

If we consider $w$ as a filter in the Fourier domain, we can express a graph convolution operation, $w * h$, where $h$ represents a signal

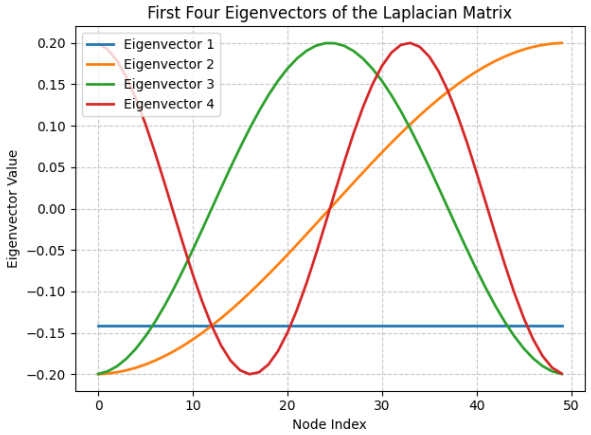


**Figure 3: Plot of the 4 smallest Eigenvectors of a line graph with 50 nodes**

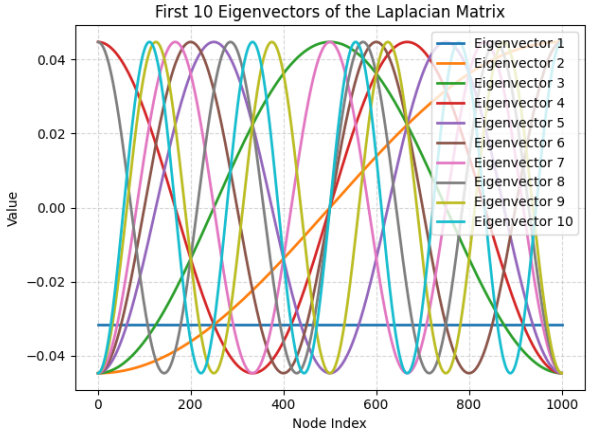


**Figure 4: Plot of the 10 smallest Eigenvectors of a line graph with 1000 nodes**

on a graph, as

$$w * h = \phi \hat{w}(\Lambda)\phi^T h = \hat{w}(L)h$$

, where $\hat{w}(\Lambda)$ is a diagonal matrix that contains in it's diagonal the values of a function that we call the spectral filter function and this matrix has the form:

$$\hat{w}(\Lambda) = \begin{bmatrix} \hat{w}(\lambda_1) & 0 & \cdots & 0 \\ 0 & \hat{w}(\lambda_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{w}(\lambda_n) \end{bmatrix}$$

(Further details can be found in Appendix).

This Laplacian eigen-decomposition and the 3 matrices product being computationally expensive, Kipf and Welling proposed a method by approximating this spectral filter with Tchebyshev polynomials of the Laplacian matrix by using learnable parameters $w_{i\in 1\ldots K}$ and We have

$$w * h = \sum_{k=0}^{K} w_k T_k(\tilde{L})h$$

with the scaled Laplacian:

$$\tilde{L} = \frac{2}{\lambda_{\max}} L - I$$

The first two Tchebythev polynomials are :

$$T_0(X) = 1$$

and

$$T_1(X) = X$$

Using a 1 order Tchebytchev approximation We have finally

$$w * h = w_0 h + w_1(L - I)h = w_0 h - w_1 \left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) h$$

.

In practice, it can be beneficial to constrain the number of parameters further to address overfitting and to minimize the number of operations (such as matrix multiplications) per layer. This leaves us with the following expression:

$$w * h \approx w \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) h$$

with a single parameter $w = w_0 = -w_1$. Note that $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ now has eigenvalues in the range $[0, 2]$.

This formulation unveils a powerful connection between the spectral properties of the Laplacian and the convolution operations on graphs, providing a deeper understanding of graph neural networks.

Repeated application of this operator can, therefore, lead to numerical instabilities and exploding/vanishing gradients when used in a deep neural network model. To alleviate this problem, we introduce the following renormalization trick:

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

, with $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

.

We can generalize this definition to a signal $X \in \mathbb{R}^{N\times C}$ with $C$ input channels (i.e., a $C$-dimensional feature vector for every node) and $F$ filters or feature maps as follows:

$$Z = \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} H W,$$

where $W \in \mathbb{R}^{C\times F}$ is now a matrix of filter parameters and $Z \in \mathbb{R}^{N\times F}$ is the convolved signal matrix. This filtering operation has complexity $O(|E|FC)$, as $A\widetilde{X}$ can be efficiently implemented as a product of a sparse matrix with a dense matrix. Finally if we add activation functions to include non linearity, the final graph convolution layer update can be written as:

$$H^{(l+1)} = \sigma\left(\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$$

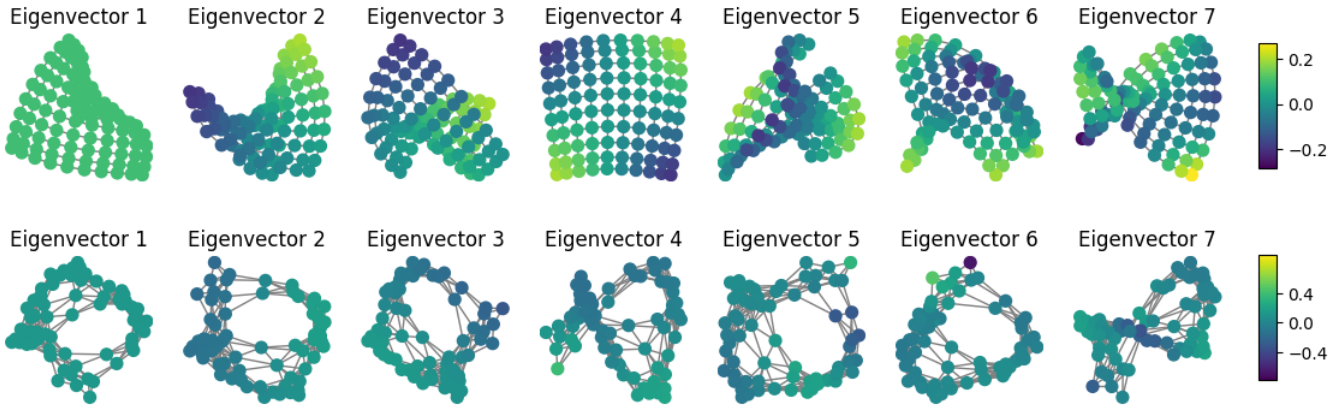


**Figure 5: Visualization of the first 7 eigenvectors for two different graphs: a 10x10 grid graph and a sensor graph. Eigenvectors are computed using the PyGSP library, and each subplot displays the values of an eigenvector. The colorbars provide a reference for the range of values in the eigenvectors. The first row corresponds to the grid graph, while the second row corresponds to the sensor graph**

where:

$H^{(l)}$ is the input feature matrix at layer,

$W^{(l)}$ is the weight matrix at layer $l$,

$\hat{A} = A + I$ is the adjacency matrix with added self-loops,

$\hat{D}$ is the degree matrix of $\hat{A}$,

$\sigma$ is the activation function.

We manage to recover the orignal vanilla GCN's aggregation function. What is important to understand in spectral convolutions is that the particularity of those convolutions actually lies in the way this spectral filter is defined. So we have seen that we can use the original convolution filter equation to define a convolution on graphs but also the method proposed by Kipef and Welling to find a much computationaly easier form. But this approximation with Tchebythev polynomials is just an idea and others approach might be used. We can also, directly learn the filter function parameters even though the convolution will still be computationally expensive to compute.

Now that we've unveiled the origin of these Laplacian eigenvectors and their connection to graph convolutions and graph signals, the question arises: why do the rows of these eigenvectors matrices effectively capture the positional encodings of nodes ? First it is intuitive as these rows are used to decompose nodes signals but other approaches have also been found such as graph cut (Appendix and [7] by Luxbourg) and random walks to justify the choice of these vectors as positional encodings. As of today there still have no proofs that demonstrate how well this eigenvectors descriibes the graph structure. Even the proofs of Luxburg linking the graph cut optimization problem with the eigenvectors of the laplacian relies on some intuitive assumptions that are not fully demonstated. So the use of those eigenvectors for PE's can still be discussed.

## 4.3 Laplacian PE results and limitations

In the paper, the autors proposed to use the representation of taking the rows of the k (a constant between 2 and 20) smallest non trivial eigevecteors matrix $\phi$ of the laplacian as positional encodings as they give the smoothest encoding. This constant k depends of the datasets and they probably tested different values to find wich one works the best on each dataset but we can debate on if this k value should be set the same for all architectures. The authors then add each of these row vectors to the graph nodes features before giving it to the MP-GCN. Something interesting is that they randomly flip the eigenvectors signs from 1 to -1 during training because both of an eigenvector and it's opposite are acutally eigenvectors so we don't know wich representation to chose. This idea to flip the sign helps the graph adapt to these ambuiguities. They proposed to use the Lapalcian PE on the datasets: CSL, CYCLE GraphTheoryProp, ZINC, AQSOL, WikiCS, PATTERN, CLUSTER and COLLAB. They use the main architectures that are GatedGCN and GIN for this test and notice that the eigenvectors improve the GNN's performance on all datasets. They tested 3 ideas for the eigenvectors: sign flipping, absolute value and k min eigenvalues and the ones that works the best is the sign flipping method. This whole experiment with PE's shows that graphs seem to actually have better performance when fed with positional encodings.

## 4.4 These ideas can be improved

We have observed the positive impact of positional encodings on enhancing the performance of Graph Neural Networks (GNNs). However, a crucial question arises: are there methods that have been explored to discover novel positional encodings that can further elevate GNN performance ? Over the past few years, various approaches have been investigated in an attempt to enhance these positional encodings in GNNs.

One avenue of exploration involves the use of Random Walk Positional Encodings. Several methodologies have been tested to

investigate their effectiveness in improving the performance of GNNs.

Another noteworthy approach is the exploration of learning positional encodings. For instance, in a study conducted by Dwivedi et al. in the paper [5], a method was proposed where learnable vectors were concatenated to the nodes features before passing them to a GNN. These vectors were initialized as random walk positional encodings (see Appendix), and the loss of the GNN related to the specifc task is regularized to ensure that these positional encodings did not deviate too far from the eigenvectors. This innovative idea addresses challenges associated with eigenvector sign flipping, mitigating the dependency of MP-GCNs on specific eigenvector signs. Consequently, this approach not only overcomes potential generalization issues but contributes to the robustness and versatility of GNNs that don't need to try to adapt to eigenvectors sign anymore.

Apart from the problem of positional encodings on graphs, GNN's suffer from another problem wich is **oversquashing**. Indeed by applying too much layers in GNN's, nodes don't know wich of the nodes information in the graphs to emphasized the most because too much information is aggregated. In the last part of the paper we see that a solution that has been tried to overcome this problem is using the graph transformers.

### 4.5 Using edge representations improve GNN's performance

To enhence the impact of **anisotropy** in GCN's, the authors uses 3 variants of GatedGCN's and GAT with no edge features and no edge representation, edge features and no edge representation and edge features and edge representation. In the first one they leverage the architecture of the gated GCN for it to be totally isotropic, in the second anisotropic approach they use a joint representation of adjacent node features to make new edge features (as they could just have intialized edges features with the true edges features of the graphs when edges do have features). In the third variant they learn edges features using nodes features and update those features with the same type of update than the nodes features. All formulas are detailed in the paper page 34 (They are not really important to understand the idea behind this part of the paper). They also did the same type of variants for GAT. They observe that adding edge features significantly improves isotropic model performance on TSP and COLLAB datasets. Explicitly maintaining edge representations enhances F1 score for TSP but degrades performance on COLLAB suggesting that the edges features are not reavelant for this task wich is a bit counter intuitive. The study suggests potential for task-specific anisotropic edge representations. They did the same type of variants for the graph sage architecture and saw that it does not improve performance on the Collab datastet. So trying to include anistropy using edges features can be sensible as it is not always easy to know wich edges feature is actually meaninful for the graph for example. More abstract representation trying to learn these edges "features" might be a better idea overall.

### 4.6 Does this benchmark hint that graph transformers might be a better approach than MP-GCN's ?

This comprehensive study emphasizes the significant enhancement of graph neural networks achieved through the integration of anisotropy, a concept that aligns intuitively. Notably, challenges such as oversquashing persist, underscoring the need for further improvement in positional encodings.

We believe that the utilization of graph transformers architecture is a promising avenue, and these ideas should be explored further. The attention mechanism in graph transformers allows direct focus on every node in the graph, independent of information from neighboring nodes, potentially alleviating the issue of oversquashing and bolstering the overall robustness of the methodology. Despite the persistent challenge of positional encodings, it is noteworthy that there is a notable increase in the number of papers addressing these issues but also on graph transformers. This collective effort means a growing understanding of the intricacies associated with graph neural networks wich is a good sign for future improvements.

At Neurips 2022, during the OGB graph level molecular tasks challenge, the top two methods employed Graph transformers type architectures, showcasing their efficacy in real-world applications. We don't affirm that graph transformers generally works better and we must do futher research on that topic as some papers contradict the effectivness of anisotropy as in the paper [9].

## 5 CONCLUSION

In this work, we examined the paper [1], an empirical benchmark of graph neural networks. We began by listing a number of criteria that, in our view, should be satisfied by a good and fair benchmark. Subsequently, we analyzed the methods employed by the authors in [1] to ensure fairness in their benchmarking of graph neural networks. Two major concerns, which we substantiated through our experiments, cast doubt on the approach taken in [1] regarding how to fairly benchmark neural network models: the choice of training parameters and the choice of hidden structure of each model. We argue that instead of using the same training parameters and hidden structure (number of hidden layers) for all networks to ensure fairness, one should be allowed to adapt these choices based on each model for optimal training, as long as they remain within the same parameter count range and are trained on the same data. We also examined how the authors used this benchmark to enhance the architectures presented in the paper. We tried to understand the authors' choice for employing Laplacian positional encodings and their connection to graph convolutions, node positions, and signals within a graph. This study led us to observe that GNN architectures lack positional information, requiring the use of positional encodings for improved performance across various tasks. Additionally, the analysis state that anisotropic GCNs generally demonstrate superior performance. We strongly believe that attention-based architectures , such as graph transformers in [3], could offer a solution to enhance both the anisotropy issues and the oversquashing phenomenon observed in MP-GCNs.

## 6 SPECIFIC CONTRIBUTIONS

- Analysis of datasets, models training and fairness (section 3) by Akedjou Achraff ADJILEYE
- Analysis of improvements using the benchmarking framework (section 4) by Melvin SEVI

## 7 REFERENCES


1 Benchmarking Graph Neural Networks, Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, Xavier Bresson
2 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.
3 Image Classification on MNIST
4 Image Classification on CIFAR-10
5 Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., & Bresson *Graph Neural Networks with Learnable Structural and Positional Representations.*
6 Dwivedi, V. P., & Bresson, X. *A Generalization of Transformer Networks to Graphs.*
7 von Luxburg, *A Tutorial on Spectral Clustering. Journal Name*
8 Kipf, T. N., & Welling, *Semi-Supervised Classification with Graph Convolutional Networks.*
9 Tailor, S. A., Opolka, F. L., & Lio; P. *Do We Need Anisotropic Graph Neural Networks?*
10 Maron, H., Ben-Hamu, H., Serviansky, H., & Lipman, Y. *Provably Powerful Graph Networks.*
11 Bresson, X., & Laurent, T. *Residual Gated Graph ConvNets.*
12 Xu, K., Hu, W., Leskovec, J., & Jegelka, S. *How Powerful are Graph Neural Networks?*


## 8 APPENDIX

**Graph Convolution with Fourier Decomposition**:

Consider a function $h$ defined on the graph. Let $\phi_k$ denote the $k$-th eigenvector of the graph. Leveraging the Fourier theorem on graphs enables us to express the graph convolution of a filter $w * h$. This operation can be decomposed using the Fourier theorem. We can get the convolution of w and h by taking the inverse fourier transform of the fourier transform of w and the fourier transform of h:

$$\begin{aligned} w * h &= \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h)) \\ &= \Phi(\Phi^T w \odot \Phi^T h) \\ &= \Phi(\hat{w} \odot \Phi^T h) \\ &= \Phi(\hat{w}(\Lambda)\Phi_1^T h) \\ &= \Phi\hat{w}(\Lambda)\Phi^T h \\ &= \hat{w}(\Phi\Lambda\Phi^T)h \\ &= \hat{w}(\Delta)h \quad (\text{where } \Delta = \Phi\Lambda\Phi^T) \end{aligned}$$

**Summary of Spectral Clustering for RatioCut Optimization**:

Let's say that we are trying to find the best partiton (with 2 sets) of the nodes of a graph that maximize the ratio cut. Here are the steps but the details can be found in the incredible paper [7] by Luxburg.

1. Objective: RatioCut Optimization (for $k = 2$)

$$\min_{A \subset V} \text{RatioCut}(A, A)$$

2. Vector Representation: We define:

$$\mathbf{f} = (f_1, \ldots, f_n)^T, \quad f_i = \begin{cases} \frac{|A|}{|A|}, & \text{if } v_i \in A, \\ -\frac{|A|}{|A|}, & \text{if } v_i \notin A. \end{cases}$$

3. Objective in Terms of Laplacian:

$$\mathbf{f}^T L \mathbf{f} = |V| \cdot \text{RatioCut}(A, A)$$

4. Orthogonality Constraint:

$$\sum_{i=1}^{n} f_i = 0$$

5. Norm Constraint:

$$\|\mathbf{f}\|_2^2 = n$$

6. Equivalent Optimization Problem (Discrete):

$$\min_{A \subset V} \mathbf{f}^T L \mathbf{f} \quad \text{subject to } \mathbf{f}^\perp \mathbf{1}, \quad \|\mathbf{f}\|_2 = \sqrt{n}$$

7. Relaxed Optimization Problem (Continuous) (ARBITRARY!!!):

$$\min_{\mathbf{f} \in \mathbb{R}^n} \mathbf{f}^T L \mathbf{f} \quad \text{subject to } \mathbf{f}^\perp \mathbf{1}, \quad \|\mathbf{f}\|_2 = \sqrt{n}$$

8. Rayleigh-Ritz Theorem: - Solution is the eigenvector corresponding to the second smallest eigenvalue of $L$.

9. Discrete Indicator Vector:

$$v_i \in A \text{ if } f_i \geq 0, \quad v_i \notin A \text{ if } f_i < 0$$

**This is actually too simple so what is actually done in practice is that we use this solution vector to deduce the cluster of each node, which is really arbitrary but intuitive.**

10. Spectral Clustering for $k = 2$: - Use k-means clustering on coordinates $f_i$ in $\mathbb{R}$. - Assign vertices to clusters based on the result.

**This is acutally super interesting because we use the previous definition of f, to make an assumption saying that we can do clustering on the values of the solution vector f instead of using the simple heuristic.**

**The case with more than two sets is a bit complicated so we won't go into this explanation. (Remember when we said that this proof is still not rigorous but it still gives a lot of intuition)**