KAGNNs: Kolmogorov-Arnold Networks meet Graph Learning

Anonymous authors
Paper under double-blind review

Abstract

In recent years, Graph Neural Networks (GNNs) have become the de facto tool for learning node and graph representations. Most GNNs typically consist of a sequence of neighborhood aggregation (a.k.a., message-passing) layers, within which the representation of each node is updated based on those of its neighbors. The most expressive message-passing GNNs can be obtained through the use of the sum aggregator and of MLPs for feature transformation, thanks to their universal approximation capabilities. However, the limitations of MLPs recently motivated the introduction of another family of universal approximators, called Kolmogorov-Arnold Networks (KANs) which rely on a different representation theorem. In this work, we compare the performance of KANs against that of MLPs on graph learning tasks. We evaluate two different implementations of KANs using two distinct base families of functions, namely B-splines and radial basis functions. We perform extensive experiments on node classification, link prediction, graph classification and graph regression datasets. Our results indicate that KANs are on-par with or better than MLPs on all tasks studied in this paper, making them viable alternatives, at the cost of some computational complexity. All code will be made available upon acceptance.

1 Introduction

Graphs are structural representations of information useful for modeling different types of data. They arise naturally in a wide range of application domains, and their abstract nature offers increased flexibility. Typically, the nodes of a graph represent entities, while the edges capture the interactions between them. For instance, in social networks, nodes represent individuals, and edges represent their social interactions. In chemo-informatics, molecules are commonly modeled as graphs, with nodes corresponding to atoms and edges to chemical bonds. In other settings, molecules can also correspond to nodes, with edges capturing their ability to bond with one another.

In many cases where graph data is available, there exist problems that cannot be solved efficiently using conventional tools (e. g., graph algorithms) and require the use of machine learning techniques. For instance, in the field of chemo-informatics, the standard approach for estimating the quantum mechanical properties of molecules leverages computationally expensive density functional theory computations (Gilmer et al., 2017). Machine learning methods could serve as a more efficient alternative to those methods. Recently, Graph Neural Networks (GNNs) have been established as the dominant approach for learning on graphs (Wu et al., 2020). Most GNNs consist of a series of message-passing layers. Within a message-passing layer, each node updates its feature vector by aggregating the feature vectors of its neighbors and combining the emerging vector with its own representation.

A lot of recent work has focused on investigating the expressive power of GNNs (Li and Leskovec, 2022). There exist different definitions of expressive power, however, the most common definition is concerned with the number of pairs of non-isomorphic graphs that a GNN model can distinguish. Two graphs are isomorphic if there exists an edge-preserving bijection between their respective sets of nodes. In this setting, a model is more expressive than another model if the former can distinguish all pairs of non-isomorphic graphs that the latter can distinguish, along with other pairs that the latter cannot (Morris et al., 2019). Furthermore,

an equivalence was established between the ability of GNNs to distinguish non-isomorphic graphs and their ability to approximate permutation-invariant functions on graphs (Chen et al., 2019). This line of work gave insights into the limitations of different models (Xu et al., 2019; Nikolentzos et al., 2020), but also led to the development of more powerful architectures (Morris et al., 2019; Maron et al., 2019; Morris et al., 2020a).

Most maximally-expressive GNN models rely on multi-layer perceptrons (MLPs) as their main building blocks, due to their universal approximation capabilities (Cybenko, 1989; Hornik et al., 1989). The theorem states that any continuous function can be approximated by an MLP with at least one hidden layer, given that this layer contains enough neurons. Recently, Kolmogorov-Arnold Networks (KANs) (Liu et al., 2024a) have emerged as promising alternatives to MLPs. They are based on the Kolmogorov-Arnold representation theorem (Kolmogorov, 1957) which states that a continuous multivariate function can be represented by a composition and sum of a fixed number of univariate functions. KANs substitute the learnable weights and pre-defined activation functions of MLPs with learnable activations and summations. The initial results demonstrate that KANs have the potential to be more accurate than MLPs in low dimensions and in settings where regularity is expected.

In this paper, we present a thorough empirical comparison of the performance between GNNs that use KANs to update node representations and GNNs that utilize MLPs to that end. Our intuition is that KANs could exploit the expected regularity underlying the message-passing paradigm to provide comparatively good performance on graph tasks. Our work is orthogonal to prior work that studies the expressive power of GNNs since we empirically compare models that are theoretically equally expressive in terms of distinguishing non-isomorphic graphs against each other, and we study the impact of the different function approximation modules (i. e., KANs or MLPs) on the models' performance. We evaluate the different GNN models on several standard node classification, link prediction, graph classification, and graph regression datasets.

The rest of this paper is organized as follows. Section 2, provides an overview of the tasks we address in this paper, as well as a description of message-passing GNNs and Kolmogorov-Arnold networks. In Section 3, we introduce the `KAGIN` (Kolmogorov-Arnold Graph Isomorphism Network) and `KAGCN` (Kolmogorov-Arnold Graph Convolution Network) models, which are variants of existing GNNs, and which leverage KANs to update node features within each layer. In Section 4, we present extensive empirical results comparing the above models with their vanilla counterparts in several tasks. Finally, Section 7 concludes the paper.

2 Background

2.1 Graph Learning Tasks

Before presenting the tasks on which we focus in this study, we start by introducing some key notation for graphs. Let $\mathbb{N}$ denote the set of natural numbers, i. e., $\{1, 2, \ldots\}$. Then, $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n \geq 1$. Let $G = (V, E)$ be an undirected graph, where V is the vertex set and E is the edge set. We denote by n the number of vertices and by m the number of edges, i. e., $n = |V|$ and $m = |E|$. Let $g \colon V \to [n]$ denote a bijective mapping from the space of nodes to set $[n]$. Let $\mathcal{N}(v)$ denote the neighbourhood of vertex v, i. e., the set $\{u \mid \{v, u\} \in E\}$. The degree of a vertex v is $\deg(v) = |\mathcal{N}(v)|$. Each node $v \in V$ is associated with a feature vector $\mathbf{x}_v \in \mathbb{R}^d$, and the feature matrix for all nodes is represented as $\mathbf{X} \in \mathbb{R}^{n \times d}$. Thus, $\mathbf{x}_v$ is equal to the $g(v)$-th row of $\mathbf{X}$.

In *node classification*, each node $v \in V$ is associated with a label y_v that represents a class. The task is to learn how to map nodes to their class labels, i. e., to learn a function f_{node} such that $f_{\text{node}}(v, G, \mathbf{X}) = y_v$. In *graph regression/classification*, the dataset consists of a collection of N graphs $G_1, \ldots, G_N$ along with their class labels/targets $y_{G_1}, \ldots, y_{G_N}$. The task is then to learn a function that maps graphs to their class labels/targets, i. e., a function f_{graph} such that $f_{\text{graph}}(G, \mathbf{X}) = y_G$, which can be discrete or continuous, for graph *classification* or graph *regression*, respectively.

The standard approach for learning such predictors (both for node- and graph-level tasks) is to first embed the nodes of the graph(s) into some vector space. That is, we aim to learn $\mathbf{H} = f_{\text{embedding}}(G, \mathbf{X}) \in \mathbb{R}^{n \times d_e}$ where d_e denotes the embedding dimension. Then, the $g(v)$-th row of matrix $\mathbf{H}$ represents the embedding of node v. Let $\mathbf{h}_v$ denote this embedding. For node-level tasks, we can use $\mathbf{h}_v$ to predict directly the

class label/target of node v. For graph-level tasks, we also need to apply a readout function on all the representations of the graph's nodes to obtain a representation $\mathbf{h}_G = f_{\text{readout}}(\mathbf{H})$ for the entire graph. One particularly desirable property of such models is permutation invariance. That is, the embedding $\mathbf{h}_G$ of a graph needs to be the same regardless of the ordering of its nodes. Indeed, these orderings do not hold any semantic meaning, and different orderings give rise to isomorphic graphs. Permutation invariance is achieved at the readout step by utilizing a permutation invariant operation over the rows of $\mathbf{H}$, such as the sum, max, or mean operators.

2.2 Graph Neural Networks

One of the most widely-used paradigms for designing such permutation invariant models is the message-passing framework (Gilmer et al., 2017) which consists of a sequence of layers and within each layer the embedding of each node is computed as a learnable function of its neighbors' embeddings. Formally, the embedding $\mathbf{h}_v^{(\ell)} \in \mathbb{R}^{d_\ell}$ at layer ℓ is computed as follows:

$$\mathbf{h}_v^{(\ell)} = \phi^{(\ell)}\left(\mathbf{h}_v^{(\ell-1)}, \bigoplus_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(\ell-1)}\right) \tag{1}$$

where $\bigoplus$ is a permutation-invariant aggregation function (e. g., mean, sum), and $\phi^{(\ell)}$ is a differentiable function (e. g., linear transformation, MLP) that combines and transforms the node's previous embedding with the aggregated vector of its neighbors.

As discussed above, we focus here on the functions that different GNN models employ to update node representations. Some GNNs use a 1-layer perceptron (i. e., a linear mapping followed by a non-linear activation function) within each neighborhood aggregation layer to update node features (Duvenaud et al., 2015; Kipf and Welling, 2017; Zhang et al., 2018). For instance, each layer of the Graph Convolutional Network (GCN) (Kipf and Welling, 2017) is defined as follows:

$$\mathbf{h}_v^{(\ell)} = \sigma\left(\mathbf{W}^{(\ell)} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{\mathbf{h}_u^{(\ell-1)}}{\sqrt{(\deg(v)+1)(\deg(u)+1)}}\right) \tag{2}$$

where σ is a non-linear activation and $\mathbf{W}^{(\ell)}$ is a trainable weight matrix.

However, the 1-layer perceptron is not a universal approximator of multiset functions (Xu et al., 2019), limiting the expressivity of the model. Thus, more recent models use MLPs instead of 1-layer perceptrons to update node representations (Balcilar et al., 2021; Dasoulas et al., 2021; Murphy et al., 2019; Nikolentzos et al., 2020). It is well-known that standard message-passing GNNs are bounded in expressiveness by the Weisfeiler-Leman (WL) test of isomorphism (Xu et al., 2019). While two isomorphic graphs will always be mapped to the same representation by such a GNN, some non-isomorphic graphs might also be assigned identical representations.

A model that can achieve the same expressive power as the WL test, given sufficient width and depth of the MLP, is the Graph Isomorphism Network (GIN) (Xu et al., 2019), which is defined as follows:

$$\mathbf{h}_v^{(\ell)} = \texttt{MLP}^{(\ell)}\left((1+\epsilon^{(\ell)}) \cdot \mathbf{h}_v^{(\ell-1)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(\ell-1)}\right) \tag{3}$$

where $\epsilon^{(\ell)}$ denotes a trainable parameter, and $\texttt{MLP}^{(\ell)}$ a trainable MLP.

The GIN model can achieve its full potential if proper weights (i. e., for the different $\texttt{MLP}^{(\ell)}$ layers and $\epsilon^{(\ell)}$) are learned, which is not guaranteed. This has motivated a series of works that focused on improving the training procedure of GNNs. For example, Ortho-GConv is an orthogonal feature transformation that can address GNNs' unstable training (Guo et al., 2022). Other works have studied how to initialize the weights of the MLPs of the message-passing layers of GNNs. It was shown that adopting the weights of converged

MLPs as the weights of corresponding GNNs can lead to performance improvements in node classification tasks (Han et al., 2023). On the other hand, there exist settings where there is no need for complex learning models. This has led to the development of methods for simplifying GNNs. This can be achieved by removing the nonlinearities between the neighborhood aggregation layers and collapsing the resulting function into a single linear transformation (Wu et al., 2019) or by feeding the node features into a neural network which generates predictions and then propagating those predictions via a personalized PageRank scheme (Gasteiger et al., 2019).

2.3 Kolmogorov-Arnold Networks

2.3.1 Principle

Presented as an alternative to the MLP, the Kolmogorov-Arnold Network (KAN) architecture has recently attracted a lot of attention in the machine learning community (Liu et al., 2024a). As mentioned above, this model relies on the Kolmogorov-Arnold representation theorem, which states that any multivariate function $f : [0, 1]^d \to \mathbb{R}$ can be written as:

$$f(\mathbf{x}) = \sum_{i=1}^{2d+1} \Phi_i \left(\sum_{j=1}^{d} \phi_{ij}(\mathbf{x}_j) \right) \tag{4}$$

where all $\Phi_\square$ and $\phi_\square$ denote univariate functions, and the sum is the only multivariate operator.

Equation 4 can be seen as a two-step process. First, a different set of univariate non-linear activation functions is applied to each dimension of the input, and then the output of those functions are summed up. The authors rely on this interpretation to define a Kolmogorov-Arnold Network (KAN) layer, which is a mapping between a space $A \subseteq \mathbb{R}^d$ and a different space $B \subseteq \mathbb{R}^{d'}$, (identical in use to an MLP layer). Such a layer consists of $d \times d'$ trainable functions $\{\phi_{ij},\ 1 \leq i \leq d',\ 1 \leq j \leq d\}$. Then, for $\mathbf{x} \in A$, we compute its image $\mathbf{x}'$ as:

$$\mathbf{x}'_i = \sum_{j=1}^{d} \phi_{ij}(\mathbf{x}_j) \tag{5}$$

Stacking two such layers, one with input dimension d and output dimension $2d + 1$, and another with input dimension $2d + 1$ and output dimension 1, we obtain Equation 4, and the derived model is a universal function approximator. This seemingly offers a complexity advantage compared to MLPs, since the number of univariate functions required to represent any multivariate function from $[0, 1]^d$ to $\mathbb{R}^{d'}$ is at most $(2d^2+d)\times d'$, whereas the universal approximation theorem for the MLP requires a possibly infinite number of neurons. However, as stated in the original paper, the behavior of such univariate functions might be arbitrarily complex (e. g., fractal, non-smooth), thus leading to them being non-representable and non-learnable.

MLPs relax the infinite-width constraint by stacking finite-width layers. Likewise, KANs relax the arbitrary complexity constraints on the non-linearities by stacking KAN layers of lower complexity. Thus, the output of a function is given by:

$$y = \texttt{KAN}(\mathbf{x}) = \Phi_L \circ \Phi_{L-1} \circ \cdots \circ \Phi_1(\mathbf{x}) \quad \text{where } \Phi_1, \ldots, \Phi_L \text{ are KAN layers.} \tag{6}$$

2.3.2 KAN Implementations and Applications

The original paper that introduced KANs uses splines (i. e., trainable piecewise-polynomial functions) as nonlinearities. This allows to retain a high expressivity for a relatively small number of parameters, at the cost of enforcing some local smoothness. A layer ℓ is thus a $d_\ell \times d_{\ell-1}$ grid of splines. The degree used for each spline (called *spline order*), as well as the number of splines used for each function (called *grid size*) are both hyperparameters of the architecture. Nonetheless, KANs allow for any function to serve as its base element. Thus, alternative implementations use different bases, such as radial-basis functions (Li, 2024), which in particular alleviate some computational bottleneck of splines.

Even though KANs were introduced very recently, they have already been applied to different problems such as in the task of satellite image classification (Cheon, 2024) and for predicting the pressure and flow

rate of flexible electrohydrodynamic pumps (Peng et al., 2024). So far, most efforts have focused on time series data (Xu et al., 2024a). For instance, KANs have been evaluated in the satellite traffic forecasting task (Vaca-Rubio et al., 2024). Furthermore, they were combined with architectures usually leveraged in time series forecasting tasks such as the LSTM Network (Genet and Inzirillo, 2024a) and the Transformer (Genet and Inzirillo, 2024b). The work closest to ours is the one reported in (Xu et al., 2024b), where the authors propose FourierKAN-GCF. This is a GNN model designed for the task of graph collaborative filtering where the feature transformation in the neighborhood aggregation layers is performed by KANs.

3 KAN-based GNN Layers

We next derive variants of the GIN and GCN models which use KANs to transform the node features instead of fully-connected layers or MLPs.

3.1 The `KAGIN` Layer

To achieve its maximal expressivity, the GIN model relies on the MLP architecture and its universal approximator property. Since KAN is also a universal function approximator, we could achieve the same expressive power using KANs in lieu of MLPs. We thus propose the `KAGIN` model, defined as:

$$\mathbf{h}_v^{(\ell)} = \texttt{KAN}^{(l)}\left((1+\epsilon)\cdot \mathbf{h}_v^{(\ell-1)} + \sum_{u\in\mathcal{N}(v)} \mathbf{h}_u^{(\ell-1)}\right) \tag{7}$$

With theoretically-sound KANs (i. e., with arbitrarily complex components), this architecture is exactly as expressive as the vanilla GIN model with infinite layer width. In practice, we employ two different families of functions to serve as the base components, leading to two different layers: `BS-KAGIN`, based on B-splines, as in the original paper; and `RBF-KAGIN`, based on radial basis functions (RBFs), which were proposed as a more computationally-efficient alternative. While these do not guarantee universal approximation, the empirical results in the original paper demonstrate the great expressive power of KANs (Liu et al., 2024a), motivating our empirical study.

3.2 The `KAGCN` Layer

GCN-based architectures have achieved great success in node classification tasks. While in our experiments we evaluate `KAGIN` on node classification datasets, the objective advantage of GCN over GIN on some of the datasets does not facilitate a fair estimation of KANs' potential in this context. To this end, we also propose a variant of the GCN model. Specifically, we substitute the parameters and ReLU function of the standard GCN (Kipf and Welling, 2017) model with a single KAN layer (defined in Equation 6) to obtain the `KAGCN` layer:

$$\mathbf{h}_v^{(\ell)} = \Phi^{(\ell)}\left(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \frac{\mathbf{h}_u^{(\ell-1)}}{\sqrt{(\deg(v)+1)(\deg(u)+1)}}\right) \tag{8}$$

where $\Phi^{(\ell)}$ denotes a single KAN layer. In the familiar matrix formulation, where $\tilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}$ is the adjacency matrix with self-loops and $\tilde{\mathbf{D}}$ the diagonal degree matrix of $\tilde{\mathbf{A}}$, the node update rule of `KAGCN` can be written as $\mathbf{H}^{(\ell)} = \Phi^{(\ell)}\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(\ell-1)}\right)$ where the different rows of $\mathbf{H}^{(\ell)}$ store the representations of the different nodes of the graph. We propose both a `BS-KAGCN` and a `RBF-KAGCN`, where Φ is based respectively on B-splines and RBFs.

4 Empirical Evaluation

In this section, we compare both `KAGIN` and `KAGCN` models against the GIN and GCN models in the following tasks: node classification, graph classification, and graph regression. The code for reproducing the results is

Table 1: Average classification accuracy (± standard deviation) and model parameters based on Luo et al.(Luo et al., 2024) using Fastkan, KAGCN, GCN on the Cora, CiteSeer, and Arxiv datasets.

Model	Cora		CiteSeer		Arxiv	
	Accuracy (%)	Params	Accuracy (%)	Params	Accuracy (%)	Params
RBF-KAGCN	**81.72 ± 0.79**	3,714,217	68.16 ± 0.61	6,258,808	71.39 ± 0.28	813,647
BS-KAGCN	80.84 ± 1.61	3,702,144	70.04 ± 1.66	4,317,952	70.58 ± 0.20	1,010,432
GCN	81.60 ± 0.40	3,262,983	**71.60 ± 0.40**	6,221,830	**71.74 ± 0.29**	2,591,784

Table 2: Hyperparameter ranges for node classification

Model	Learning Rate	Hidden Layers	Hidden Dim.	Grid Size	Spl. Order
GIN	$[1e-5, 1e-2]$	$[1, 4]$	$[8, 1024]$	NA	NA
BS-KAGIN/KAGCN	$[1e-5, 1e-2]$	$[1, 4]$	$[2, 64]$	$[1, 8]$	$[1, 4]$
RBF-KAGIN/KAGCN	$[1e-5, 1e-2]$	$[1, 4]$	$[2, 64]$	$[2, 32]$	NA

available at `anonymized`. All models are implemented with PyTorch (Paszke et al., 2019). For KAN layers, we rely on publicly available implementations for B-splines [1] and for RBF [2]. For a given task, all models have consistent architectures, differing as little as possible.

4.1 Node classification

Datasets To evaluate the performance of GNNs with KAN layers in the context of node classification, we use 7 well-known datasets of varying sizes and types, including homophilic (Cora, Citeseer (Kipf and Welling, 2017) and Ogbn-arxiv (Hu et al., 2020)) and heterophilic (Cornell, Texas, Wisconsin, Actor) networks. The homophilic networks are already split into training, validation, and test sets, while the heterophilic datasets are accompanied by fixed 10-fold cross-validation indices.

Experimental setup. We perform two types of experiments. First, we compare our KAGCN model on the homophilic datasets to the GCN proposed in the state of the art approach in (Luo et al., 2024). The results are shown in Table 1. One can observe that the classical GCN outperforms the KAN-based model in 2 out of 3 datasets, albeit with a narrow difference and confidence overlaps in all of the comparisons. Note that current setting is built to favor the classical GNNs, as it relies on techniques such as dropout, jumping knowledge, batch normalization and preliminary linear transformations. Hence, although the MLP-based GCN enjoys a plethora of assistive techniques examined in the past decades, the KAN-based counterpart exhibits a close performance. We should note here that these assistive techniques have not been developed yet for KAN-based architectures, and our results aim at motivating the development of such techniques.

Our second approach compares the vanilla versions of the neural architectures when we replace the MLP with KAN, as delineated in sections 3. We only use message passing layers, skip connections and an output layer. We do not utilize dropout, batchnorm, jumping knowledge etc. The reason we constrain ourselves to the bare minimum is to isolate and clarify the effect of KAN on the model's performance. In other words, we aspire to compare the classical MLP-based GNNs as if they were in the same nascent stage as their KAN-based counterparts. Having said that, we perform an auxiliary experiment afterwards, to position the KAGCN in the SOTA and clarify its current utility.

We then evaluate the 10 models on the test set and report the average accuracy. For the heterophilic datasets, we report the average accuracy across the 10 folds. In order for all models to exploit the same quantity of information, we fix the number of message-passing layer dataset-wise, with 2 for Cora and CiteSeer, 3 for Texas, Cornell, Wisconsin, and Ogbn-arxiv and 4 for Actor.

[1] `https://github.com/Blealtan/efficient-kan`
[2] `https://github.com/ZiyaoLi/fast-kan`

Table 3: Average classification accuracy (± standard deviation) of the `KAGIN`, GIN, `KAGCN`, and GCN models on the 7 node classification datasets.

Model	Cora	Citeseer	Ogbn-arxiv	Cornell	Texas	Wisconsin	Actor
GIN	67.48±6.53	61.04±6.42	35.68±12.9	41.70±1.70	45.14±6.32	43.24±2.68	26.43±0.86
`BS-KAGIN`	76.28±0.98	63.42±2.06	30.53±5.59	**48.92**±1.08	**62.30**±0.23	50.78±1.54	28.00±0.37
`RBF-KAGIN`	72.34±3.74	44.26±12.0	21.77±13.2	47.49±1.85	59.92±1.88	51.73±1.51	**28.91**±0.40
GCN	75.00±4.11	62.35±3.87	40.60±6.75	41.27±1.96	55.32±3.27	52.02±1.12	28.64±0.24
`BS-KAGCN`	**78.99**±0.80	**66.81**±1.25	52.67±4.07	48.73±1.58	58.70±0.52	**56.18**±1.09	27.07±0.27
`RBF-KAGCN`	63.59±7.10	48.00±8.32	**54.56**±3.99	45.86±2.10	58.43±1.21	53.35±1.38	27.10±0.31

Table 4: Hyperparameter ranges for link prediction

Model	Learning Rate	Hidden Layers	Hidden Dim.	Grid Size	Spl. Order
GIN	$1e-2, 1e-3, 1e-4$	$2, 4$	$[16, 512]$	NA	NA
`BS-KAGIN`/`KAGCN`	$1e-2, 1e-3$	$2, 4$	$[16, 64]$	$[8]$	$[2, 3]$
`RBF-KAGIN`/`KAGCN`	$1e-2, 1e-3$	$2, 4$	$[16, 512]$	$[2, 16]$	NA

For every dataset and model, we tune the values of the hyperparameters using the Optuna package (Akiba et al., 2019). For each parameterization, we train 5 models and select the parameterization that yields the lowest mean validation loss (cross-entropy) across all 5 models. We set the number of iterations of Optuna equal to 100 trials with a TPE Sampler. We use early stopping with a patience of 50. Once the best hyperparameter values are found, we evaluate the models on the test set. For the homophilic networks, we initialize and train 10 different models with the best hyperparameters (using 10 different random seeds). The ranges for hyperparameters are different for each model. Indeed, for the same number of "neurons", a KAN-based layer will have a much larger number of parameters than an MLP-based layer (see Appendix B.1 for examples). In order to avoid giving an unfair advantage to KAN-based layers, we thus allowed the MLP layers to be much larger, so that the largest possible model in all cases had around the same number of parameters. The hyperparameters ranges are given in table 4.

Results. The results are given in Table 3. We see that KAN-based variants beat MLP-based models on all datasets, while the B-splines-based variant outperforms both other models on 5/7 datasets. We also observe that `RBF-KAGIN` seems to exhibit a high instability, especially on Citeseer and Ogbn-arxiv, where its performance is much lower than that of other models, and its variance a lot higher. We notice that the simpler GCN-based models are more efficient than the more complex GIN-based models on all three homophilic datasets, while the opposite is true on heterophilic datasets. This is coherent with the inherent complexity discrepency between both families of datasets (Zhu et al., 2021).

Overall, B-splines are more effective for both graph learning models, the difference being clear mostly in homophilic networks. Overall we contend that KAN has a positive impact on both, the GIN and the GCN architecture. Moreover, the introduction of KAN does not alleviate the inherent disadvantage of the model e. g., GCN's low accuracy on heterophilic networks, however, it improves the models in the majority of the datasets.

4.2 Link Prediction

Datasets and Experimental Setup. In this setting, we focused on the task of link prediction. We use two datasets, Cora and CiteSeer, following the implementation and protocol provided in Li et al. (2023); Mao et al. (2024). We performed a grid-search over learning rate, hidden dimension, number of hidden layers, grid size and spline order, each time training models on 10 splits. We then selected the model with lowest validation loss to obtain the best configuration, and reported its test ROC-AUC (over all 10 splits).

Results. Table 5 displays the results on the link prediction task. We see that the standard GCN outperforms the other GCN-based architecture on both datasets, but `RBF-KAGIN` is the overall best performer on Cora.

Table 5: AUC for link prediction of the `KAGIN` and GIN models on the link-prediction datasets.

	Cora	**CiteSeer**		**Cora**	**CiteSeer**
GIN	94.72 ± 0.25	95.07 ± 0.37	GCN	94.90 ± 0.20	**96.07** ± 0.23
`BS-KAGIN`	94.68 ± 0.49	95.42 ± 0.31	`BS-KAGCN`	94.31 ± 0.55	94.26 ± 0.55
`RBF-KAGIN`	**95.60** ± 0.44	95.69 ± 0.79	`RBF-KAGCN`	94.76 ± 0.81	93.91 ± 1.58

Table 6: Hyperparameter ranges for node classification

Model	Learning Rate	Hidden Layers	Hidden Dim.	Dropout	Grid Size	Spl. Order
GIN	$[1e-4, 1e-2]$	$[1, 4]$	$[2, 512]$	$[0, 0.9]$	NA	NA
`BS-KAGIN/KAGCN`	$[1e-4, 1e-2]$	$[1, 4]$	$[2, 64]$	$[0, 0.9]$	$[2, 32]$	$[1, 4]$
`RBF-KAGIN/KAGCN`	$[1e-4, 1e-2]$	$[1, 4]$	$[2, 64]$	$[0, 0.9]$	$[2, 16]$	NA

4.3 Graph Classification

Datasets. In this set of experiments, we compare the `KAGIN` models against GIN on standard graph classification benchmark datasets (Morris et al., 2020b). We experiment with the following 7 datasets: (1) MUTAG, (2) DD, (3) NCI1, (4) PROTEINS, (5) ENZYMES, (6) IMDB-B, (7) IMDB-M. The first 5 datasets come from bio- and chemo-informatics, while the last 2 are social interaction datasets.

Experimental setup. We follow the experimental protocol proposed in (Errica et al., 2020). We kept our architecture simple. First, a number of message-passing layers extract node representation, whose model is either an MLP (resp. a KAN). We then use sum pooling to obtain graph representations. Finally, an MLP (resp. a KAN) with same architecture as those found in the message-passing layer, is used as a classifier. A softmax is used for predicting class probabilities. Thus, we perform 10-fold cross-validation, while within each fold a model is selected based on a 90%/10% split of the training set. We use the pre-defined splits provided in (Errica et al., 2020). We use the Optuna package to select the model that achieves the lowest validation cross-entropy loss over 100 iterations. For a fair comparison, we set the number of message-passing layers of all architectures to a fixed value for each dataset. On MUTAG, PROTEINS, IMDB-B and IMDB-M, we set the number of layers to 2. On DD, we set it to 3. On ENZYMES, we set it to 4 and finally, on NCI1, we set it to 5. We train each model for $1,000$ epochs (early stopping with a patience of 20 epochs) by minimizing the cross entropy loss. We use the Adam optimizer for model training (Kingma and Ba, 2015). We apply batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) to the output of each message-passing layer. The ranges for hyperparameters search are given in table **??**. Once the best hyperparameters are found for a given split, we train 3 different models on the training set of the split and evaluate them on the test set of the split. This yields 3 test accuracies for this split, and we compute their average to obtain the performance for this split.

Results. Table 7 illustrates the average classification accuracies and the corresponding standard deviations of the three models on the different datasets. We observe that GIN and both `KAGIN` models achieve similar levels of performance on all datasets except ENZYMES. In particular, `RBF-KAGIN` and `BS-KAGIN` both have the best accuracy on 2 datasets each, and GIN on 1 dataset. Regarding the GCN variants, we observe that the KAN-based variants now have a significant advantage over the linear GCN, confirming that a single KAN layer is more expressive than a simple fully-connexted layer. `BS-KAGCN` and `RBF-KAGCN` obtain the best overall accuracy on 1 dataset each, while the vanilla GCN performs often significantly poorly.

These experiments show that, on a graph-level task, which requires significantly more power than node-level tasks, KAN-based models seem to perform better than their MLP-based counterparts. While this difference is not significant among GIN-like architectures, it appears that KAN-based GCNs benefit from the higher expressivity in a single layer, and attain performances much closer to their GIN-like counterparts, sometimes even outperforming them.

Table 7: Average classification accuracy (± standard deviation) of the KAGIN and GIN models on the 7 graph classification datasets.

	MUTAG	DD	NCI1	PROTEINS	ENZYMES	IMDB-B	IMDB-M
GIN	85.26 ± 7.30	74.78 ± 4.65	78.32 ± 2.49	71.01 ± 2.81	53.28 ± 1.23	73.13 ± 3.97	**50.27** ± 4.01
BS-KAGIN	85.08 ± 6.07	75.37 ± 3.37	**79.44** ± 1.67	74.09 ± 3.60	42.72 ± 8.94	**73.37** ± 4.03	49.91 ± 3.47
RBF-KAGIN	**85.48** ± 5.36	**75.66** ± 3.08	78.33 ± 1.81	73.29 ± 3.56	41.17 ± 8.32	71.97 ± 3.81	48.76 ± 4.53
GCN	56.34 ± 15.1	63.24 ± 3.82	60.42 ± 1.01	60.46 ± 2.06	22.06 ± 5.58	68.70 ± 3.63	46.24 ± 4.83
BS-KAGCN	75.55 ± 8.14	73.31 ± 4.45	71.59 ± 3.31	**75.55** ± 3.62	57.67 ± 8.03	70.83 ± 3.95	50.00 ± 4.41
RBF-KAGCN	73.52 ± 11.6	68.30 ± 4.53	63.95 ± 7.80	74.18 ± 3.64	**59.39** ± 8.03	72.60 ± 4.61	48.53 ± 4.03

Table 8: Average MAE (± standard deviation) of the KAGIN and GIN models on graph regression.

	ZINC-12K	QM9
GIN	0.4131 ± 0.0215	0.0969 ± 0.0017
BS-KAGIN	**0.3000** ± 0.0332	**0.0618** ± 0.0007
RBF-KAGIN	0.3026 ± 0.0663	0.0778 ± 0.0023

4.4 Graph Regression

Datasets. We experiment with two molecular datasets: (1) ZINC-12K (Irwin and Shoichet, 2005), and (2) QM9 (Ramakrishnan et al., 2014). ZINC-12K consists of $12,000$ molecules. The task is to predict the constrained solubility of molecules, an important property for designing generative GNNs for molecules. The dataset is already split into training, validation and test sets ($10,000$, $1,000$ and $1,000$ graphs in the training, validation and test sets, respectively). QM9 contains approximately $134,000$ organic molecules. Each molecule consists of Hydrogen, Carbon, Oxygen, Nitrogen, and Flourine atoms and contain up to 9 heavy (non Hydrogen) atoms. The task is to predict 12 target properties for each molecule. The dataset was divided into a training, a validation, and a test set according to a 80%/10%/10% split.

Experimental setup. We perform grid search to select values for the different hyperparameters. We choose the number of hidden layers from $\{2, 3, 4\}$ for GIN and $\{1, 2, 3\}$ for KAN. We use different ranges for each model was because we wanted to limit the size advantage of KAN against MLP, since a layer with the same width of KAN has more parameters than an MLP one). For both models we search a learning rate from $\{10^{-3}, 10^{-4}\}$. For GIN, we choose the hidden dimension size from $\{32, 64, 128, 256, 512, 1024\}$, while for KAGIN, we choose it from $\{4, 8, 16, 32, 64, 128, 256\}$. For KAGIN, we also select the grid size from $\{1, 3, 5, 8, 10\}$ and for BS-KAGIN the spline order from $\{3, 5\}$. To produce graph representations, we use the sum operator. The emerging graph representations are finally fed to a 2-layer MLP (for GIN) or a 2-layer KAN (for KAGIN) which produces the output. We set the batch size equal to 128 for all models. We train each model for $1,000$ epochs by minimizing the mean absolute error (MAE). We use the Adam optimizer for model training (Kingma and Ba, 2015). We also use early stopping with a patience of 20 epochs. For ZINC-12K, we also use an embedding layer that maps node features into 100-dimensional vectors. We choose the configuration that achieves the lowest validation error. Once the best configuration is found, we run 10 experiments and report the average performance on the test set. For both datasets and models, we set the number of message-passing layers to 4. On QM9, we performed a joint regression of the 12 targets.

Results. The results are shown in Table 8. We observe that on both considered datasets, both KAGIN variants significantly outperform the GIN model. Note that these datasets are significantly larger (in terms of the number of samples) compared to the graph classification datasets of Table 7. The results also indicate that BS-KAGIN outperforms RBF-KAGIN on both ZINC-12K and QM9. However, the difference in performance between the two models is very small. Thus, it appears that both B-splines and RBFs are good candidates as KANs' base components in regression tasks. In addition, we should note that the BS-KAGIN model offers

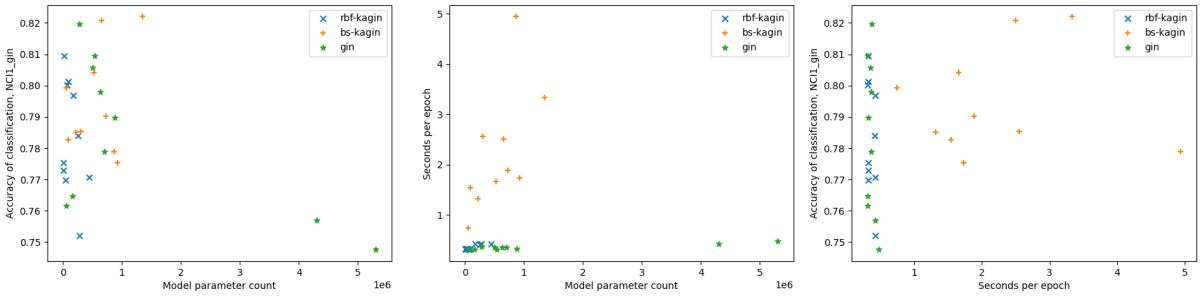

Figure 1: Parameters count/size/performance comparison, GIN, NCI1

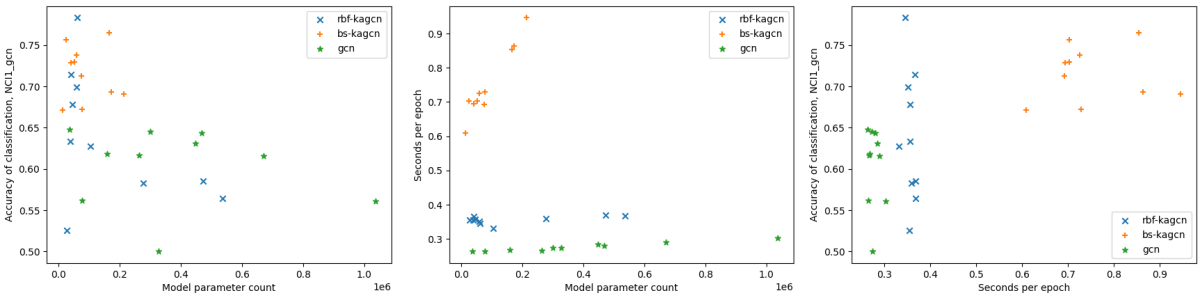

Figure 2: Parameters count/size/performance comparison, GCN, NCI1

an absolute improvement of approximately 0.11 and 0.03 in MAE over GIN. Those improvements suggest that KANs might be more effective than MLPs in regression tasks.

5 Analysis of Size and Time

5.1 Selected models

Following the above results, and it is legitimate to wonder whether the performance of the KAN models comes from a larger size than their MLP-based counterparts. We show in this section that it is not the case. Using the graph-classification models trained for the results provided in 4.3, we display here three relations: the size-to-performance relation, the size-to-training time relation and the training time-to-performance relation.

Figures 2 and 4 show these relations for datasets NCI1 and IMDB-MULTI respectively. We also show in 9 and 10 the numerical values associated to these graphs. The rest of the graph classification datasets are provided in App. B.1. A first thing that we can see from both figures is that, for the same parameter count, B-splines-based KANs are significantly slower than both other architectures. RBF-based KANs are much faster, showing that this implementation succeeds in alleviating the computational bottleneck induced by the sequential computation necessary for B-splines. Finally, MLPs are the fastest for equal parameter counts. While these results were expected, we notice in both the tables and figures that `RBF-KAGIN`models end up significantly smaller than their MLP-based counterparts, while maintaining equal or better performance, and about equal training time. On the other hand, the B-Splines based method is significantly slower, by sometimes a 5 fold factor. This establishes the RBF-based architecture as the best compromise between scalability and performance, although further work could be made in studying the performance of very small KAN-based against MLP-based models.

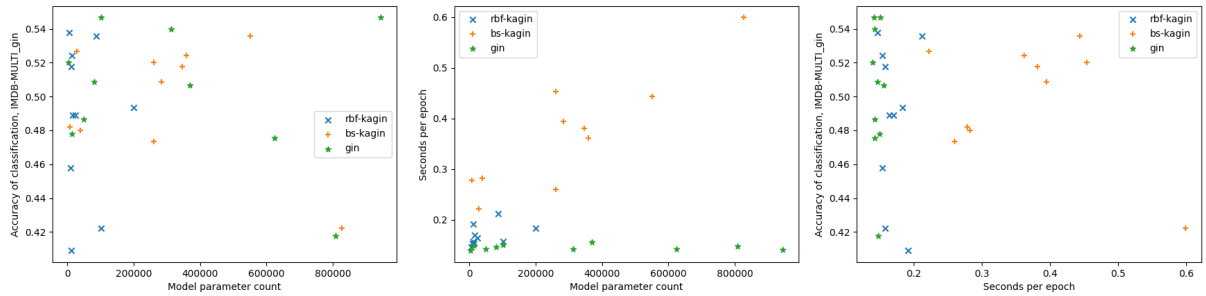

Figure 3: Parameters count/size/performance comparison, GIN, IMDB-MULTI

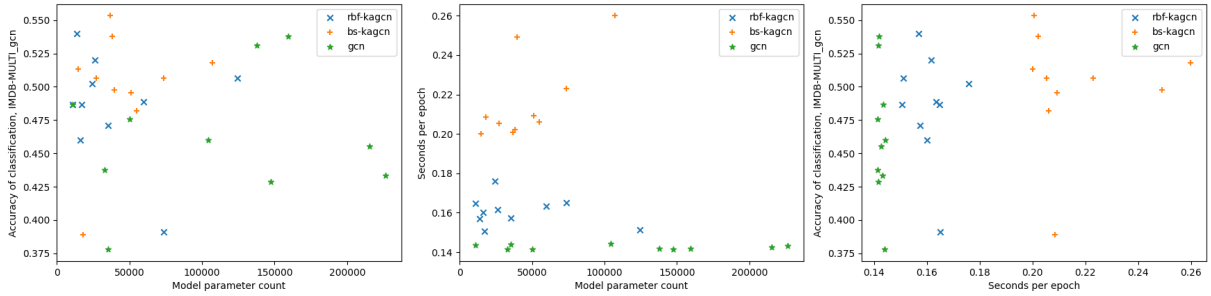

Figure 4: Parameters count/size/performance comparison, GCN, IMDB-MULTI

Table 9: Time and parameter sizes for selected models, NCI1

Model	Accuracy	Parameter count	time per epoch (s)
BS-KAGIN	0.794 ± 0.016	572338.4	2.220
RBF-KAGIN	0.783 ± 0.017	139168.0	0.358
GIN	0.783 ± 0.024	1337369.8	0.359
BS-KAGCN	0.716 ± 0.031	90196.5	0.752
RBF-KAGCN	0.640 ± 0.075	166960.5	0.356
GCN	0.604 ± 0.046	379634.0	0.277

Table 10: Time and parameter sizes for selected models, IMDB-MULTI

Model	Accuracy	Parameter count	time per epoch (s)
BS-KAGIN	0.499 ± 0.033	296418.8	0.367
RBF-KAGIN	0.488 ± 0.043	48705.1	0.169
GCN	0.503 ± 0.038	330987.6	0.146
BS-KAGIN	0.500 ± 0.042	46052.3	0.216
RBF-KAGIN	0.485 ± 0.038	40118.5	0.161
MLP	0.462 ± 0.046	111911.4	0.143

6 Discussion and Perspectives

Prominent application domains. KANs, both in our work and in the recent literature, seem to be a promising alternative to MLPs for regression tasks. They rely on combinations of continuous functions which could explain their improved performance for continuous outputs. Moreover, this property and their empirical performance in regression tasks renders them an ideal candidate for physics-inspired architectures (Karniadakis et al., 2021), which address problems of high importance, from climate forecasting (Kashinath et al., 2021) to atomic simulations (Batzner et al., 2022) and rely heavily on graph-based architectures.

Explainability matters. Recent works on KANs argue that their potential for explainability is higher than that of MLPs with post-hoc methods (e.g. Shapley values, gradient-based) Liu et al. (2024a), since closed-form symbolic formulas can be drawn from the former Liu et al. (2024b), providing complete overview of the underlying computations. Hence, KAN-based graph models could potentially prove more friendly to practitioners from application fields, such as biology and physics, who tend to be skeptical of the black-box approach.

GNNs for fixed graphs inspired from KANs. Each KAN architecture consists of a set of neurons. Each one of those neurons is connected to other neurons. Thus, KANs can be naturally modeled as graphs where nodes represent neurons and edges represent the connections between them. Each layer of a KAN is actually a complete bipartite graph, while a KAN has learnable activation functions on the edges of its associated graph. There actually exist datasets that contain multiple graphs which share the same graph structure, but differ in the node and edge features. Two examples of such datasets are the MNIST and CIFAR10 graph classification datasets (Dwivedi et al., 2023). Since the graph structure is fixed (and identical for all graphs), we could associate each edge of the graph with one or more learnable activation functions (shared across graphs), similar to what KANs do. This would allow the model to learn different functions for each edge of the graph, leading potentially to high-quality node and graph representations.

Advantages and future work. Finally, we discuss potential advantages that KANs might have over MLPs, and leave their investigation for future work. First, their ability to accurately fit smooth functions could prove relevant on datasets where variables interact with some regular patterns. Second, their interpretability could be leveraged to provide explanations on learned models, giving insights into the nature of interactions among entities. Finally, a thorough study of the effect of the different hyperparameters could be pursued, allowing to fully exploit the splines or radial basis functions while retaining small networks.

7 Conclusion

In this paper, we have investigated the potential of Kolmogorov-Arnold networks in graph learning tasks. Since the KAN architecture is a natural alternative to the MLP, we developed two GNN architectures, `KAGCN` and `KAGIN`, respectively analogous to the GCN and GIN models. We then compared those architectures against each other in both node- and graph-level tasks. For link prediction, as well as graph and node classification, there does not appear to be a clear winner. For graph regression, our results indicate that KAN has an advantage over MLP, with the B-spline variant proving superior in terms of performance. Nonetheless, this gain comes at the cost of computational efficiency. In this regard, the RBF-based vairant offers a slightly lesser performance improvement, but proves much more efficient in terms of training time.

This paper shows that such KAN-based GNNs are valid alternatives to the traditional MLP-based models. We thus believe that these models deserve the attention of the graph machine-learning community, and that future work should be carried out to identify their specific advantages and weaknesses, allowing for an enlightened choice between KANs and MLPs based on the task and context at hand.

References

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*, pages 1263–1272, 2017.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1): 4–24, 2020.

Pan Li and Jure Leskovec. The expressive power of graph neural networks. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 63–98, 2022.

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 33rd AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019.

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*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. K-hop graph neural networks. *Neural Networks*, 130:195–205, 2020.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably Powerful Graph Networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 2156–2167, 2019.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *Advances in Neural Information Processing Systems*, pages 21824–21840, 2020a.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024a.

Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk*, 114(5):953–956, 1957.

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*, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 4438–4445, 2018.

Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning,*, pages 599–608, 2021.

George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In *Proceedings of the 29th International Conference on International Joint Conferences on Artificial Intelligence*, pages 2126–2132, 2021.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *Proceedings of the 36th International Conference on Machine Learning,*, pages 4663–4673, 2019.

Kai Guo, Kaixiong Zhou, Xia Hu, Yu Li, Yi Chang, and Xin Wang. Orthogonal graph neural networks. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, pages 3996–4004, 2022.

Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. MLPInit: Embarrassingly simple gnn training acceleration with mlp initialization. In *Proceedings of the 11th International Conference on Learning Representations*, 2023.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6861–6871, 2019.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Ziyao Li. Kolmogorov-arnold networks are radial basis function networks, 2024. URL `https://arxiv.org/abs/2405.06721`.

Minjong Cheon. Kolmogorov-arnold network for satellite image classification in remote sensing. *arXiv preprint arXiv:2406.00600*, 2024.

Yanhong Peng, Miao He, Fangchao Hu, Zebing Mao, Xia Huang, and Jun Ding. Predictive modeling of flexible ehd pumps using kolmogorov-arnold networks. *arXiv preprint arXiv:2405.07488*, 2024.

Kunpeng Xu, Lifei Chen, and Shengrui Wang. Kolmogorov-arnold networks for time series: Bridging predictive power and interpretability. *arXiv preprint arXiv:2406.02496*, 2024a.

Cristian J Vaca-Rubio, Luis Blanco, Roberto Pereira, and Màrius Caus. Kolmogorov-arnold networks (kans) for time series analysis. *arXiv preprint arXiv:2405.08790*, 2024.

Remi Genet and Hugo Inzirillo. Tkan: Temporal kolmogorov-arnold networks. *arXiv preprint arXiv:2405.07344*, 2024a.

Remi Genet and Hugo Inzirillo. A temporal kolmogorov-arnold transformer for time series forecasting. *arXiv preprint arXiv:2406.02486*, 2024b.

Jinfeng Xu, Zheyu Chen, Jinze Li, Shuo Yang, Wei Wang, Xiping Hu, and Edith C-H Ngai. Fourierkan-gcf: Fourier kolmogorov-arnold network–an effective and efficient feature transformation for graph collaborative filtering. *arXiv preprint arXiv:2406.01034*, 2024b.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 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.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic gnns are strong baselines: Reassessing gnns for node classification. *arXiv preprint arXiv:2406.08993*, 2024.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. Graph neural networks with heterophily. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11168–11176, 2021.

Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. In *Neural Information Processing Systems NeurIPS, Datasets and Benchmarks Track*, 2023.

Haitao Mao, Juanhui Li, Harry Shomer, Bingheng Li, Wenqi Fan, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Revisiting link prediction: A data perspective. *The Twelfth International Conference on Learning Representations*, 2024.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020b.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15 (1):1929–1958, 2014.

John J Irwin and Brian K Shoichet. Zinc- a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 45(1):177–182, 2005.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1(1):1–7, 2014.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Karthik Kashinath, M Mustafa, Adrian Albert, JL Wu, C Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, R Wang, Ashesh Chattopadhyay, A Singh, et al. Physics-informed machine learning: case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A*, 379(2194):20200093, 2021.

Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):2453, 2022.

Ziming Liu, Pingchuan Ma, Yixuan Wang, Wojciech Matusik, and Max Tegmark. Kan 2.0: Kolmogorov-arnold networks meet science. *ArXiv*, abs/2408.10205, 2024b. URL `https://api.semanticscholar.org/CorpusID:271903197`.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *Advances in Neural Information Processing Systems*, pages 22326–22340, 2022.

A Further Graph Classification Experiments

Besides the graph classification experiments that are presented in the main paper, we also experimented with the Peptides-func dataset, a graph classification dataset from the Long Range Graph Benchmark (Dwivedi et al., 2022). The dataset consists of $15,535$ graphs where the average number of nodes per graph is equal to 150.94. We evaluated GIN, `BS-KAGIN` and `RBF-KAGIN` on this dataset. We performed a grid search where for all models we chose the learning rate from $\{0.001, 0.0001\}$, the number of hidden layers of MLPs and KANs from $\{1, 2, 3\}$ and the hidden dimension from $\{64, 128\}$. We set the number of neighborhood aggregation layers to 5 for all models. For both `BS-KAGIN` and `RBF-KAGIN`, grid size was chosen from $\{3, 5\}$, and for `BS-KAGIN`, spline order was chosen from $\{3, 5\}$. Each experiment was run 4 times with 4 different seeds and Table 11 illustrates the average precision achieved by the three models.

Table 11: Average Precision (± standard deviation) of the `KAGIN` and GIN models on the Peptides-func dataset.

	Peptides-func
GIN	55.01 ± 0.75
`BS-KAGIN`	**58.71** ± 1.02
`RBF-KAGIN`	57.68 ± 2.05

We observe that `BS-KAGIN` and `RBF-KAGIN` outperform GIN on the Peptides-func dataset. Both models offer an absolute improvement of more than 2.5% in average precision over GIN. Thus, even in tasks where long-range interactions between nodes need to be captured, for a fixed receptive field, KAN layers can help the models better capture those interactions.

B Time/Performance/Accuracy analysis

Figures 6 to 14 and tables 12 to 16 regarding the time/size/accuracy analysis on the graph classification task.

B.1 Training Times - Graph Classification

We give in Table 17 the training time per epoch for different configurations of the `BS-KAGIN`, `RBF-KAGIN` and GIN models on a graph-level task. We observe that for a given number of message-passing layers and hidden dimension size, GIN is the fastest, followed by `RBF-KAGIN`. `BS-KAGIN` is noticeably slower than the others. It suffers doubly: its parameter count increases much quicker than both other models for same hyperparameters; but even for similar numbers of parameters, `BS-KAGIN` is by far the slowest. Spline order seems to have a larger impact on running time than grid size. This shows that RBFs do alleviate some of the computational bottlenecks of the B-splines from the original KAN paper. Table 18 in Appendix C shows similar observations on a node-level task.

Table 12: Time and parameter sizes for selected models, IMDB-BINARY

Model	Accuracy	Parameter count	time per epoch (s)
`BS-KAGIN`	0.734 ± 0.038	148236.3	0.225
`RBF-KAGIN`	0.720 ± 0.036	168383.6	0.129
GIN	0.731 ± 0.038	1172520.5	0.110
`BS-KAGCN`	0.708 ± 0.037	39053.5	0.150
`RBF-KAGCN`	0.726 ± 0.044	37322.5	0.109
GCN	0.687 ± 0.034	118826.3	0.099

Table 13: Time and parameter sizes for selected models, ENZYMES

Model	Accuracy	Parameter count	time per epoch (s)
BS-KAGIN	0.427 ± 0.085	450136.6	0.336
RBF-KAGIN	0.412 ± 0.079	115346.8	0.061
GIN	0.533 ± 0.117	1623033.2	0.048
BS-KAGCN	0.577 ± 0.076	83636.3	0.103
RBF-KAGCN	0.594 ± 0.053	125854.4	0.050
GCN	0.221 ± 0.053	280040.0	0.039

Table 14: Time and parameter sizes for selected models, DD

Model	Accuracy	Parameter count	time per epoch (s)
BS-KAGIN	0.754 ± 0.032	169417.4	1.206
RBF-KAGIN	0.757 ± 0.029	147005.1	0.236
GIN	0.748 ± 0.044	168957.4	0.144
BS-KAGCN	0.733 ± 0.042	110278.6	0.596
RBF-KAGCN	0.683 ± 0.043	110899.2	0.185
GCN	0.632 ± 0.036	342023.2	0.225

Table 15: Time and parameter sizes for selected models, MUTAG

Model	Accuracy	Parameter count	time per epoch (s)
BS-KAGIN	0.851 ± 0.058	236235.8	0.056
RBF-KAGIN	0.855 ± 0.051	62530.0	0.020
GIN	0.853 ± 0.069	848213.0	0.014
BS-KAGCN	0.755 ± 0.077	19496.1	0.022
RBF-KAGCN	0.735 ± 0.110	47769.4	0.017
GCN	0.563 ± 0.143	72072.2	0.012

Table 16: Time and parameter sizes for selected models, PROTEINS_full

Model	Accuracy	Parameter count	time per epoch (s)
BS-KAGIN	0.741 ± 0.034	321367.7	0.472
RBF-KAGIN	0.733 ± 0.034	51498.8	0.096
MLP	0.710 ± 0.027	942781.0	0.073
BS-KAGCN	0.756 ± 0.034	37688.7	0.133
RBF-KAGCN	0.742 ± 0.035	48950.4	0.078
GCN	0.605 ± 0.020	8199.8	0.061

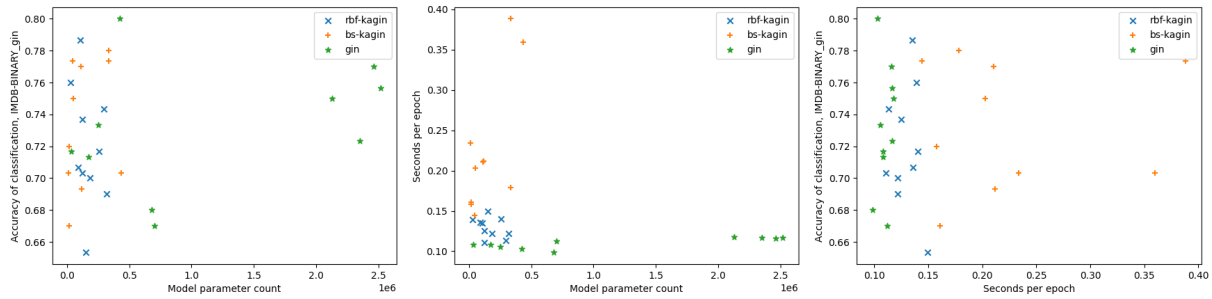

Figure 5: Parameters count/size/performance comparison, GIN, IMDB-BINARY

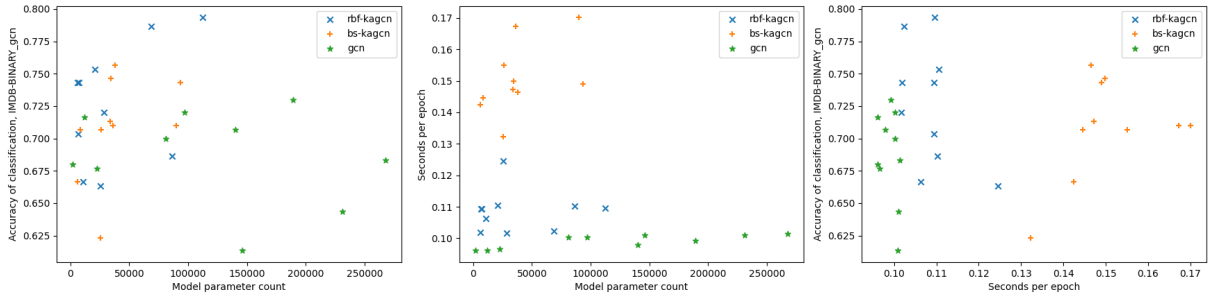

Figure 6: Parameters count/size/performance comparison, GCN, IMDB-BINARY

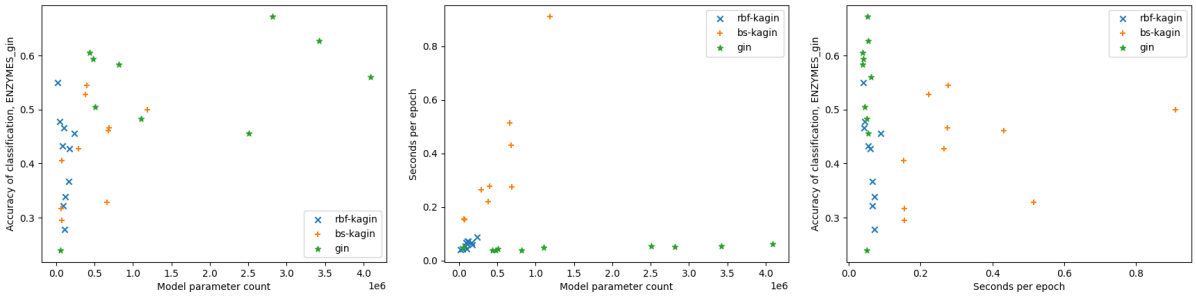

Figure 7: Parameters count/size/performance comparison, GIN, ENZYMES

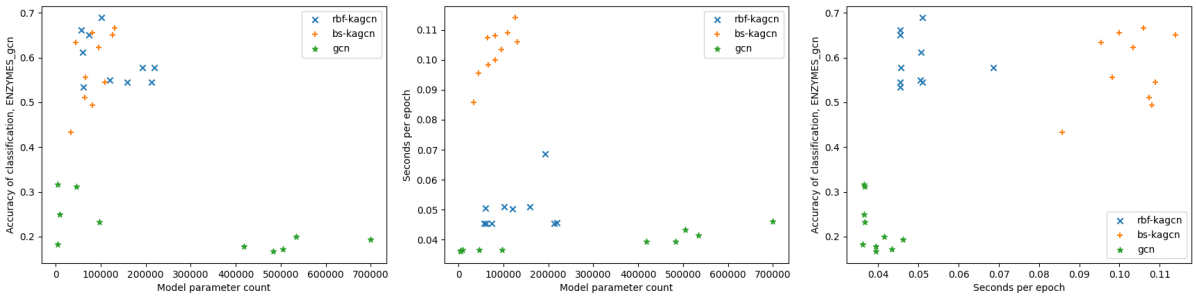

Figure 8: Parameters count/size/performance comparison, GCN, ENZYMES

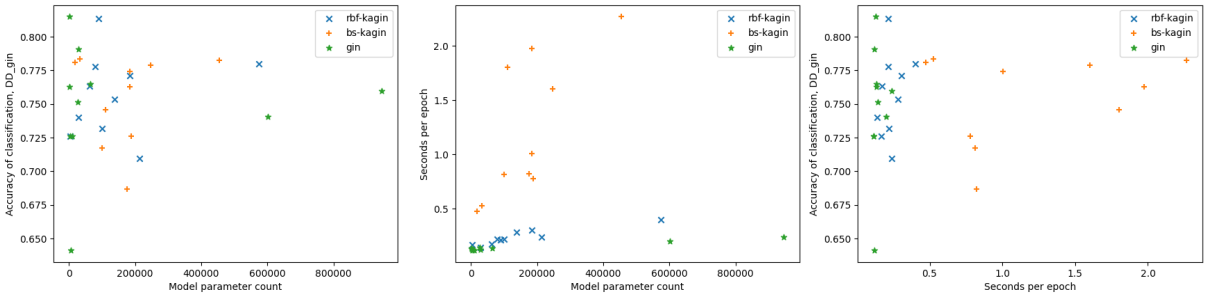

Figure 9: Parameters count/size/performance comparison, GIN, DD

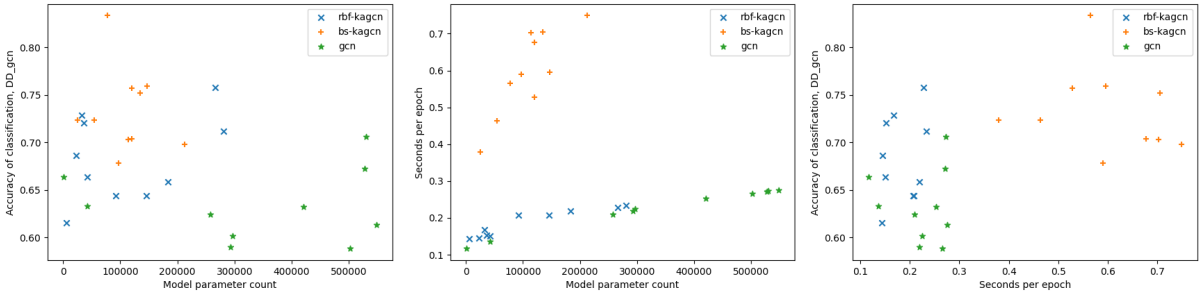

Figure 10: Parameters count/size/performance comparison, GCN, DD

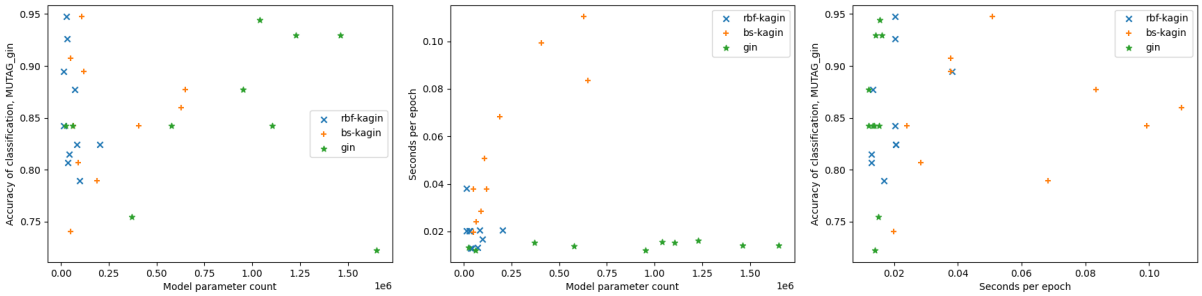

Figure 11: Parameters count/size/performance comparison, GIN, MUTAG

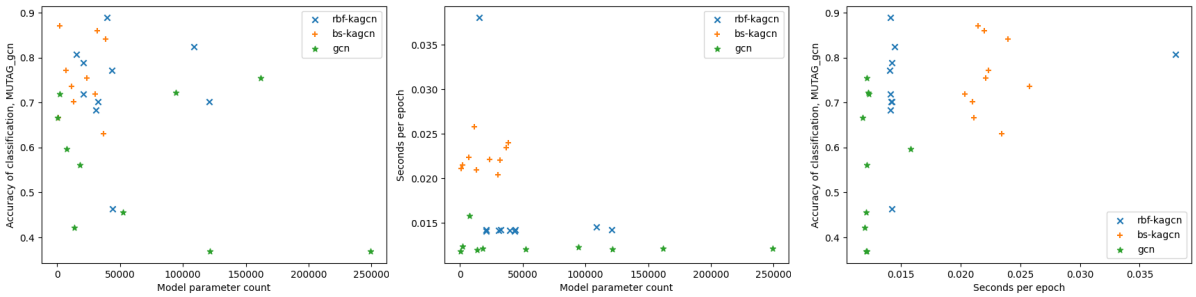

Figure 12: Parameters count/size/performance comparison, GCN, MUTAG

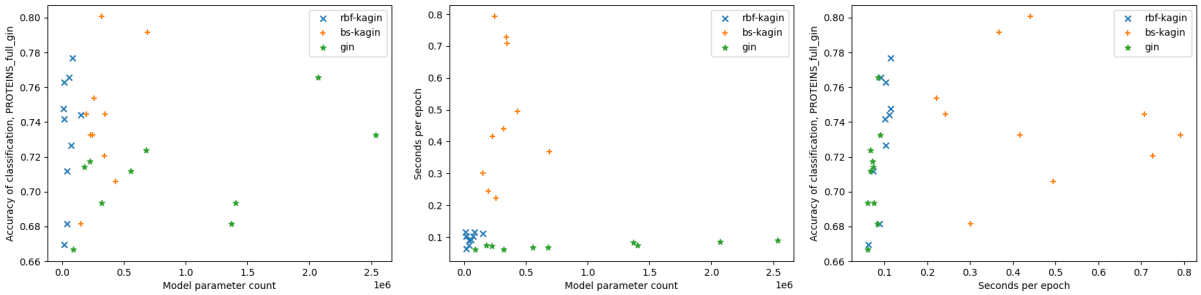

Figure 13: Parameters count/size/performance comparison, GIN, PROTEINS_full

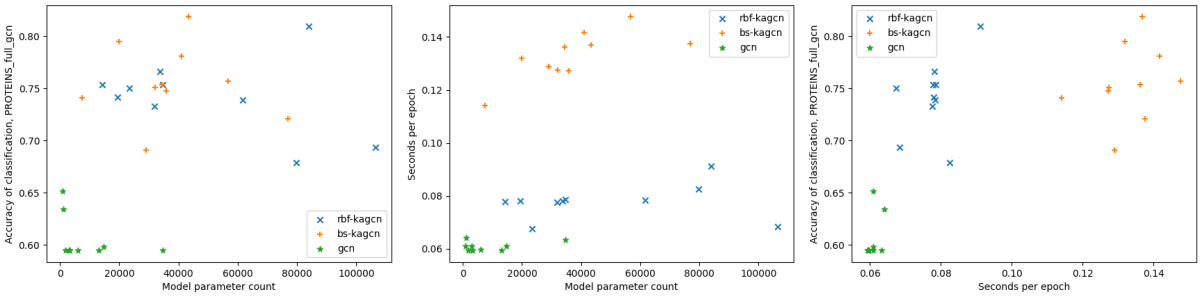

Figure 14: Parameters count/size/performance comparison, GCN, PROTEINS_full

Table 17: Training time per epoch on NCI1 dataset. 5 message-passing layers. batch size set to 128.

Architecture	Hidden Dimension	Hidden Layers	Grid Size	Spline Order	# Parameters	Training time (s/epoch)
GIN	256	2	NA	NA	670722	0.303
GIN	256	4	NA	NA	1465346	0.390
GIN	512	2	NA	NA	2652162	0.541
GIN	512	4	NA	NA	5814274	0.951
RBF-KAGIN	256	1	1	NA	822872	0.386
RBF-KAGIN	256	1	4	NA	1639274	0.599
RBF-KAGIN	256	3	1	NA	3191408	0.783
RBF-KAGIN	256	3	4	NA	6367142	1.577
RBF-KAGIN	512	1	1	NA	3218520	0.807
RBF-KAGIN	512	1	4	NA	6424170	1.441
RBF-KAGIN	512	3	1	NA	12674160	2.198
RBF-KAGIN	512	3	4	NA	25317030	4.429
BS-KAGIN	256	1	1	1	1091072	0.446
BS-KAGIN	256	1	1	4	1907456	2.964
BS-KAGIN	256	1	4	1	1907456	0.720
BS-KAGIN	256	1	4	4	2723840	4.240
BS-KAGIN	256	3	1	1	4236800	0.937
BS-KAGIN	256	3	1	4	7412480	9.703
BS-KAGIN	256	3	4	1	7412480	1.963
BS-KAGIN	256	3	4	4	10588160	14.572

C Training Times - Node Classification

Table 18: Training time per epoch on the node-classification task (Cora dataset, 3 MP-layers).

Architecture	Hidden Dimension	Hidden Layers	Grid Size	Spline Order	# Parameters	Training time (s/epoch)
GCN	256	NA	NA	NA	378934	0.003
GCN	1024	NA	NA	NA	1485622	0.005
BS-KAGCN	256	NA	1	1	1514684	0.017
BS-KAGCN	256	NA	1	4	2650697	0.160
BS-KAGCN	256	NA	4	1	2650697	0.031
BS-KAGCN	256	NA	4	4	3786710	0.222
BS-KAGCN	512	NA	1	1	2989244	0.020
BS-KAGCN	512	NA	1	4	5231177	0.179
BS-KAGCN	512	NA	4	1	5231177	0.039
BS-KAGCN	512	NA	4	4	7473110	0.254
RBF-KAGCN	256	NA	1	NA	1142524	0.016
RBF-KAGCN	256	NA	4	NA	2278543	0.031
RBF-KAGCN	1024	NA	1	NA	4462588	0.029
RBF-KAGCN	1024	NA	4	NA	8916367	0.057
GIN	256	2	NA	NA	867335	0.007
GIN	256	4	NA	NA	1130503	0.008
GIN	1024	2	NA	NA	5042183	0.020
GIN	1024	4	NA	NA	9240583	0.033
BS-KAGIN	256	1	1	1	1514684	0.017
BS-KAGIN	256	1	1	4	2650697	0.158
BS-KAGIN	256	1	4	1	2650697	0.032
BS-KAGIN	256	1	4	4	3786710	0.224
BS-KAGIN	256	3	1	1	3990528	0.025
BS-KAGIN	256	3	1	4	6983424	0.229
BS-KAGIN	256	3	4	1	6983424	0.051
BS-KAGIN	256	3	4	4	9976320	0.334
BS-KAGIN	512	1	1	1	2989244	0.021
BS-KAGIN	512	1	1	4	5231177	0.184
BS-KAGIN	512	1	4	1	5231177	0.041
BS-KAGIN	512	1	4	4	7473110	0.261
BS-KAGIN	512	3	1	1	10078208	0.046
BS-KAGIN	512	3	1	4	17636864	0.354
BS-KAGIN	512	3	4	1	17636864	0.089
BS-KAGIN	512	3	4	4	25195520	0.499
RBF-KAGIN	256	1	1	NA	1142524	0.017
RBF-KAGIN	256	1	4	NA	2278543	0.032
RBF-KAGIN	256	3	1	NA	3002487	0.023
RBF-KAGIN	256	3	4	NA	5995401	0.046
RBF-KAGIN	1024	1	1	NA	4462588	0.03
RBF-KAGIN	1024	1	4	NA	8916367	0.057
RBF-KAGIN	1024	3	1	NA	21429879	0.091
RBF-KAGIN	1024	3	4	NA	42838665	0.185