

Comparative Analysis of Graph Neural Networks and Kernel Methods for Molecular HIV Inhibition Prediction

Candidate IDs: 51348, 42371, 37984, 50691

Department of Statistics

London School of Economics and Political Science

Abstract

Molecular property prediction is crucial in drug discovery, particularly in identifying compounds that inhibit HIV replication. For this purpose we compare two graph-based learning paradigms, graph neural networks (GNNs) and graph kernel methods. We evaluated GIN and GCN variants under various pooling and normalization strategies to optimize ROC-AUC performance. In parallel, we benchmark kernel-based approaches, which, despite their interpretability, are underexplored in this context. Using the OGB-MolHIV dataset, we show that both approaches yield similar results and argue for the complementary strengths of deep learning and kernel methods.

Keywords: molecular property prediction, graph neural networks, kernel methods, HIV inhibition, cheminformatics

1 Introduction

This report focuses on the binary classification task of predicting whether a chemical compound acts as an HIV inhibitor. This is an important classification task that can accelerate the discovery of new molecules to cure diseases. Instead of testing whether a molecule will inhibit HIV experimentally, we can rely on machine learning methods to quickly produce results. While GNNs have shown remarkable success in this task, kernel approaches offer mathematically grounded alternatives that are underexplored for these molecular graphs.

In this paper we use the OGB-MolHIV benchmark dataset to conduct a comparative study of the aforementioned methods in terms of predictive performance, generalization, and computational efficiency. Molecules are modeled as graphs, with atoms as nodes and bonds as edges, enabling the use of advanced machine learning techniques on non-Euclidean data. We show that while graph kernels do not reach the highest performance on the OGB leaderboard (84 % AUC), they perform similarly to strong neural network-based baselines such as GCN, GIN, and GatedGCN, with typical performances around 75% AUC.

There have been recent advances in both graph kernels and GNN’s. In terms of graph kernels, these advances are well summarized on Borgwardt et al. (2020). In this book, the authors argue that graph kernels are a form of “feature engineering” on graphs. With this reasoning, they provide a taxonomy of graph kernels: bag of structures, information propagation, extensions to incorporate continuous covariates and extensions beyond the \mathcal{R} -convolution framework. Bag-of-structures kernels can be thought of as counting a specific kind of substructure in the graph, information propagation kernels consist of random walk kernels and the different instances of the message passing framework. Extensions to con-

tinuous graphs allow for the use of kernels in a wider variety of graphs. Finally there are kernels that are not based on the \mathcal{R} -convolution framework, like Weisfeiler-Lehman (WL) Optimal Assignment(OA), which are designed to reduce the overfitting observed in the message passing framework. Interestingly, we find that simple bag-of-structures kernels like the shortest path kernel perform as well as the baseline WL framework, and better than the more sophisticated WL-OA kernel.

Graph Neural Networks (GNNs) have emerged as a powerful approach for learning representations from graph-structured data Li et al. (2016); Hamilton et al. (2017); Kipf and Welling (2017); Velickovic et al. (2018); Xu et al. (2019). GNNs operate under the message passing paradigm, where node embeddings are iteratively updated by aggregating information from neighboring nodes. After k iterations, each node’s representation reflects its k -hop neighborhood, integrating both local topology and feature information. To obtain graph-level predictions, these node embeddings are pooled using techniques such as simple summation Xu et al. (2019), attention-based aggregation Lee et al. (2019), or sequence-based methods like Set2Set Vinyals et al. (2016).

Many architectural variants of GNNs have been proposed to enhance representational capacity and training stability. GCNs Kipf and Welling (2017) use weighted neighborhood averaging, while GINs Xu et al. (2019) replace this with sum aggregation followed by MLPs to capture finer structural distinctions. Gated GCNs Bresson and Laurent (2017) introduce gating and residual paths to address vanishing gradients in deeper networks. Furthermore, recent designs such as virtual node augmentation Hu et al. (2020b) explicitly inject a globally connected node to promote long-range information flow. While these advances yield strong empirical results, the design of GNN architectures remains largely heuristic, and the understanding of the theoretical limitations of their expressivity and generalization, incomplete.

To systematically compare graph kernels and graph neural networks for molecular property prediction, we organise our study as follows. Section 2 describes the dataset and provides essential details on node, edge, and graph-level features. In Section 3, we formulate the problem and present the core model families: GNN architectures, as well as the most widely used kernel, the WL framework. Section 4 presents comparative results for all models, both neural and kernel-based. It then discusses the expressive power of pooling functions and the impact of normalization strategies for GNN methods. In Section 5, we conclude with reflections on model limitations and outline future extensions to improve generalization and scalability.

2 Data Description

We use the OGBG-MolHIV dataset from the Open Graph Benchmark (OGB), consisting of 41,127 molecular graphs derived from MoleculeNet (Wu et al., 2018) and processed with RDKit (Landrum, 2006). Each undirected graph represents a molecule, where nodes denote atoms and edges represent chemical bonds. On average, graphs contain 25 nodes and 27 edges.

Node and edge features are categorical and encoded using `AtomEncoder` and `BondEncoder` into low-dimensional continuous embeddings suitable for GNNs. The dataset follows OGB’s standard split: 32,941 training, 4,116 validation, and 4,070 test graphs. All splits exhibit

similar statistics, with node counts ranging from 25–28, edge counts from 54–61, average node degree around 2.1–2.2, and low clustering coefficients, reflecting sparse local connectivity.

Table 1: Statistics of the OGBG-MolHIV dataset splits.

Split	Samples	Nodes	Edges	Degree	Clustering
Train	32,941	25.3 ± 12.0	54.1 ± 26.1	2.1 ± 0.8	0.001 ± 0.028
Validation	4,116	27.8 ± 12.8	61.1 ± 27.8	2.2 ± 0.8	0.003 ± 0.042
Test	4,070	25.3 ± 12.5	55.6 ± 27.1	2.2 ± 0.7	0.004 ± 0.058

As shown in Figure 1, the number of nodes per graph ranges from 10 to 60, with long-tailed distributions extending beyond 100 nodes. While average node and edge counts are consistent across splits, training graphs have slightly heavier tails. Node degrees are skewed toward 1–3, typical of sparsely connected organic molecules. Most atoms exhibit low clustering coefficients, though some cyclic structures show higher values. All distributions are plotted on a log scale, highlighting their long-tailed nature and consistency across splits.

Figure 2 shows a strong class imbalance: 39,684 Non-Toxic vs. 1,443 Toxic molecules (3.5%). Although both classes have similar median atom counts, Toxic molecules tend to be slightly larger, with broader atom count distributions and occasional large outliers in both classes.

3 Problem formulation and solution

In this section, we review foundational concepts in Graph Neural Networks (GNNs) and Graph Kernels relevant to molecular property prediction. We describe the main features of the different GNN architectures we implement, highlighting the differences amongst them. Finally we describe the key definitions of graph kernels. In particular we focus on the Weisfeiler-Lehman framework, one of the most intuitive instances of the message passing kernels.

3.1 Graph Neural Networks

Let a graph be denoted by $G = (V, E)$, where V is the set of nodes¹ and $E \subseteq V \times V$ is the set of edges. Each node $v \in V$ is associated with an initial feature vector $\mathbf{x}_v \in \mathbb{R}^d$.² In molecular graphs, nodes represent atoms and edges represent chemical bonds.

A GNN operates under a message-passing framework, where each node updates its representation by aggregating messages from its neighbors over multiple iterations (or layers). At each iteration $t \in \{1, \dots, T\}$, the node representation $\mathbf{h}_v^{(t)}$ is computed as follows:

$$\mathbf{m}_v^{(t)} = \text{AGGREGATE}^{(t)} \left(\left\{ \mathbf{h}_u^{(t-1)} : u \in \mathcal{N}(v) \right\} \right), \quad \mathbf{h}_v^{(t)} = \text{UPDATE}^{(t)} \left(\mathbf{h}_v^{(t-1)}, \mathbf{m}_v^{(t)} \right)$$

1. Nodes and vertices are the same and used interchangeably during this paper.
2. In all GNN models, bond features \mathbf{e}_{uv} are encoded via a shallow **BondEncoder** network, and node features \mathbf{x}_v are encoded with **AtomEncoder** embeddings. This is explained in more detail in Subsection 4.1.

where $\mathcal{N}(v)$ denotes the neighbors of node v , and $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. The AGGREGATE function must be permutation-invariant (e.g., sum, mean, max), and the UPDATE function is typically implemented using a multi-layer perceptron (MLP) or gating mechanism.

After T layers, the final node representations $\mathbf{h}_v^{(T)}$ can be aggregated into a graph-level representation using a readout function:

$$\mathbf{h}_G = \text{READOUT} \left(\left\{ \mathbf{h}_v^{(T)} : v \in V \right\} \right)$$

where READOUT is often implemented as a summation, mean, or a more sophisticated pooling operator like attention-based pooling or Set2Set.

This framework allows GNNs to capture both local and global structural information, which is critical for tasks such as molecular property prediction.

3.2 Model Architectures

Graph Convolutional Networks (GCN) (Kipf and Welling, 2017) Our GCN layer is based on the Graph Convolutional Network (GCN) formulation, adapted to include bond embeddings and a root embedding term to stabilize node updates. The node update rule is:

$$\mathbf{h}_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{(d_v + 1)(d_u + 1)}} \text{ReLU}(\mathbf{W}\mathbf{h}_u^{(k-1)} + \mathbf{e}_{uv}) + \frac{1}{d_v + 1} \text{ReLU}(\mathbf{W}\mathbf{h}_v^{(k-1)} + \mathbf{b}),$$

where d_v denotes the node degree plus self-loop adjustment, \mathbf{W} is a learnable weight matrix, \mathbf{b} is the root embedding, and \mathbf{e}_{uv} is the bond embedding.

Graph Isomorphism Networks (GIN) (Xu et al., 2019) Our GIN layer follows the Graph Isomorphism Network (GIN) design with two key modifications: (i) we incorporate bond (edge) features \mathbf{e}_{uv} , and (ii) we apply batch normalization within the MLP. Formally, the node update rule is:

$$\mathbf{h}_v^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon) \cdot \mathbf{h}_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} \text{ReLU}(\mathbf{h}_u^{(k-1)} + \mathbf{e}_{uv}) \right),$$

where $\mathbf{h}_v^{(k)}$ denotes the node embedding at layer k , \mathbf{e}_{uv} is the encoded bond embedding, and ϵ is a learnable scalar parameter.

Residual Gated Graph Convolutional Networks (GatedGCN) (Bresson and Laurent, 2017) Our Gated GCN architecture follows the Gated Graph Convolutional Network formulation, where node representations are updated using gated recurrent units to better capture complex dependencies over multiple hops. Unlike traditional GCNs, the Gated GCN incorporates recurrent message passing across layers and allows for more expressive node updates. Formally, the update rule is:

$$\mathbf{h}_v^{(k)} = \text{GRU}^{(k)} \left(\mathbf{h}_v^{(k-1)}, \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(k-1)} \right),$$

where $\mathbf{h}_v^{(k)}$ is the node embedding at layer k , and the GRU combines the previous node embedding $\mathbf{h}_v^{(k-1)}$ with the aggregated messages from neighbors. Each GatedGCN layer thus refines node representations based on gated aggregation mechanisms.

Virtual Node Augmentation (Gilmer et al., 2017; Hu et al., 2020b) To enable global information flow, we add a synthetic virtual node connected to all other nodes in the graph. It aggregates messages from the entire graph and broadcasts a shared update at each layer, improving communication across distant nodes. The virtual node is updated via a residual MLP with batch normalization and ReLU, and its embedding is injected into each node before the next message-passing step. This mechanism alleviates the limited receptive field of standard GNNs and enhances graph-level representation learning.

3.3 Kernel Methods

To compare our baseline GNN’s to graph kernels, we tried using a combination of graph kernels and Support Vector Classifiers. Below, we define a kernel on a graph. Then, we describe one of our best performing graph kernels, the Weisfeiler–Lehman(WL) framework, in detail.³

Kernel *Given a non-empty set \mathcal{X} , we say that a function $k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a kernel if there exists a Hilbert space \mathcal{H} and some map $\phi: \mathcal{X} \rightarrow \mathcal{H}$ that satisfies*

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$$

for all $x, x' \in \mathcal{X}$.

Our set \mathcal{X} will be the graphs in our dataset. These labeled graphs are defined by a dictionary (of all possible labels) Σ_V and a labeling function: $l_V: V \rightarrow \Sigma_V$, which maps from the vertices to the dictionary values.

Two of our best-performing methods, WL and Neighborhood Hashing, belong to the message passing framework and so are very similar conceptually. Despite Neighborhood-Hashing having slightly higher performance on the test set, we explain the WL framework, which implements the iterative label refinement in a more intuitive manner.

In the WL framework, we start with the labels of the original graphs ($h = 0$) and iteratively refine them so they include information on their neighbors. A refinement step is defined as follows:

WL update *Given refinement step h , a vertex v a neighborhood around a vertex $N(v)$ and a hashing function $f(\cdot)$ the WL update consists of*

$$l_V(v)_WL^{(h+1)} = f(l_V(v)_WL^{(h)}, \{l_V(v')_WL^{(h)} | v' \in N(v)\})$$

for all $v \in V$.

That is, for a given set of labels from your neighbors, as well as your own label, you will get a unique number that identifies the particular combination of your label and the set of neighbors’ labels.

3. This section is largely adapted from Borgwardt et al. (2020). Due to space limitations we only describe WL framework in detail, but good descriptions of the remaining kernels we tried can be found in the aforementioned book.

Repeating this iteration h times gives u a sequence of graphs with different vertex labels for each iteration:

WL sequence *Given an undirected graph $G = (V, E)$ with a label function $l_v(\cdot)$ and $h \in \mathbb{N}$, the WL relabeling yields a sequence (G_0, \dots, G_h) , where*

$$G_h := (V, E, l_V(\cdot))_{WL}^{(h)}$$

Given the sequence, we can define the WL kernel:

WL kernel *Let G and G' be two undirected graphs with node labels defined over the same alphabet. Given a well-defined base kernel κ and $h \in \mathbb{N}$, the WL kernel is defined as*

$$k_{WL}^{(h)}(G, G') := \sum_{i=0}^h \kappa(G_i, G'_i)$$

where G_i is an element of the WL sequence.

In words, we use a base kernel to compute the similarity between the two graphs at each label refinement step and then we sum them up to compute a global measure of similarity across steps. As a base kernel we use a Vertex Histogram kernel.⁴ This kernel counts the instances a label from a dictionary (all distinct labels in G and G') is in a given graph G . The similarity of the two graphs is then computed via the dot product amongst label counts of the two graphs.

4 Numerical Results and Interpretation

This section discusses all implementation details that were introduced to boost performance, in both GNN and kernel based methods. Thereafter, we present a detailed evaluation of five graph neural network models⁵ and five Kernel methods⁶ trained on the OGB-MolHIV dataset for binary classification.

4.1 Implementation Details

In terms of GNN-based models we experiment with different pooling and batch normalization operators. The categorical features of both nodes and edges are encoded into a continuous space using learnable encoders that are trained jointly with the GNN. Kernel methods required some tuning due to the large computational requirements. We strategically subsampled the data, keeping all positive training graphs and a random sample of negative examples.

To enhance model stability and generalization, we include dropout after ReLU activations and apply various normalization techniques (BatchNorm, LayerNorm, InstanceNorm, GraphNorm) after each convolutional layer. We also experimented with different pooling operators, as described in Section 5. We also used `BondEncoder` and `AtomEncoder` for edges and nodes respectively. These encoders are learnable and provide a real-valued representation of the original features, which are categorical and thus cannot be handled successfully by GNN's.

4. The formal definition is in Borgwardt et al. (2020) but it is not too intuitive.

5. The GNN models are: GIN, Virtual Node GIN, GCN, Gated GCN, and GCN with Virtual Node.

6. Shortest Path, WL framework, Neighborhood Hashing, WL-OA, Core based framework.

For kernels methods, we ignore all edge attributes and all but one vertex level attribute -atomic number-, which we will treat as the label of a given vertex. We also strategically subsampled the training data into 5,000 graphs - keeping all positive (minority) samples while randomly selecting from negative (majority) samples. This preserved class representation while reducing computational requirements⁷, however, the validation and test sets were kept intact for proper evaluation. Finally, we employed SVMs with precomputed kernel matrices. We used probabilistic SVM estimates computed via Platt scaling Platt (1999) - fitting a sigmoid function to the SVM decision values for calibrated probability estimates. Despite the increased computational cost, this calibration was essential for the evaluation of AUC.

4.2 Results

Table 2 presents results from selected graph kernel and GNN-based methods. These results are from the best GNN methods -best pooling and batch normalization results - and the hypertuned kernel-SVM methods.⁸ While kernel methods are slightly outperformed by our GNN approaches -GIN + Virtual Node is our best overall method with 0.785 test AUC-they both provide similar performances while likely focusing on different features of the data. All our methods fall slightly short of state-of-the-art GNN approaches on the OGB leaderboard (84% AUC). This suggests significant room for improvement in our methods.

Interestingly, different methods utilizing fundamentally different graph information achieve remarkably similar test results. WL and Neighborhood hashing are very similar to GNN but in our case they were only trained using the atomic number of the vertex, not with the (continously) embedded data with which the GNN-based models were trained. Furthermore, within kernel methods, information propagation kernels (WL and Neighborhood Hashing) perform similarly to the Shortest Path kernel, despite the latter ignoring label information entirely. This points in the direction of using ensemble methods to aggregate the different information outputted by different methods. The top performing method on the OGB leaderboard, Zhang et al. (2023), follows such an approach.

To conclude, long training times are no guarantee of good test performance. While in GNN based methods⁹, longer training networks do have slightly better test performance, this positive relation is completely broken for kernels. The shortest path kernel is trained for 2 minutes to achieve good test accuracy, WL framework achieves 75.00% test accuracy in just 4 minutes, while Neighborhood Hashing reaches 76.51% in 24 minutes. Finally, the computationally intensive WL-OA (158 minutes) has one of the worst performances on the test set.

-
- 7. The RAM in our computers was not enough to support a 30000×30000 array.
 - 8. Notably, when we conducted hyperparameter tuning on the validation set for the kernels, the test AUC decreased compared to default parameters. However, we report only hypertuned kernel-SVM results.
 - 9. These trends are further contextualized by the theoretical complexity breakdown provided in Table 7, which highlights the computational overhead associated with various pooling strategies in GNNs.

Model	Validation AUC	Test AUC	Implementation Time (min)
GNN-based Methods			
GCN	0.807	0.752	48
GCN + Virtual Node	0.820	0.763	42
Gated GCN	0.802	0.773	28
GIN	0.837	0.781	66
GIN + Virtual Node	0.809	0.785	65
Kernel-based Methods			
Shortest Path	0.771	0.752	2
WL Framework	0.831	0.750	4
Neighborhood Hashing	0.822	0.765	24
WL-OA	0.847	0.726	158
Core-Based Framework	0.793	0.716	—

Table 2: Validation and Test ROC-AUC along with training time for GNN and kernel-based models. Bold denotes best performance across all models in a given metric.

4.3 Effect of Pooling Operators

The empirical results in Table 6 support the theoretical distinctions outlined in Appendix 5. While sum pooling is theoretically the most expressive due to its injectivity over multisets, its empirical performance is mixed. GIN performs best with sum pooling (0.837, 0.781), aligning with its design philosophy (Xu et al., 2019), but GCN performs worse with sum compared to mean and attention (0.757, 0.762), suggesting that theoretical expressiveness does not always yield better results and is architecture-dependent. Max pooling consistently underperforms, likely due to its loss of frequency and distributional information. Attention-based pooling, which generalizes mean with learned node-level weights, performs robustly and achieves the best test AUC for GCN (0.804), likely due to its ability to prioritize informative nodes. Set2Set offers moderate gains (e.g., GIN+VN) but its higher cost and instability may limit its utility.

Overall, while sum pooling is theoretically appealing, attention and mean pooling offer more consistent practical benefits, particularly for GCN-based models. These results highlight the importance of treating pooling strategy as a tunable hyperparameter.

4.4 Effect of Normalization Strategies

Normalization was crucial for stable and efficient GNN training. As shown in Table 5, different normalization strategies had varying impacts on model performance across architectures. **BatchNorm** consistently delivered strong, reliable performance across architectures, balancing convergence speed and representation quality, even though it was originally designed for i.i.d. data. LayerNorm and InstanceNorm exhibited more variable results. LayerNorm showed limited benefits, likely due to its insensitivity to structural variations within graph neighborhoods. InstanceNorm performed better for some models (e.g., GCN) by smoothing

optimization within each graph, but its tendency to suppress structural information may limit expressiveness in graphs with subtle functional patterns, like molecular data.

GraphNorm proved a promising alternative, balancing optimization stability and expressiveness. Its learnable shift after graph-wise normalization preserves feature variance and aids convergence. It was especially helpful for stabilizing deeper models like Gated GCN, though not always outperforming BatchNorm. This aligns with theory Cai et al. (2021) suggesting that graph-level normalization improves generalization in tasks with fine-grained topological patterns.

5 Conclusion

We implement different GNN and kernel based methods to a molecule prediction issue that is key for drug discovery. Both our approaches yield similar results (75-78 % AUC) which are slightly below the state-of-the-art approaches (84 % AUC).

During this project we encountered some bottlenecks, one was the technical compatibility between libraries.¹⁰ Another key challenge was navigating the vast design space of Graph Neural Networks (GNNs).¹¹ Finally, while we evaluated widely used architectures such as GCN, GIN, and GatedGCN, we were unable to explore more expressive architectures¹² due to resource constraints.¹³

Future research on GNN models- could explore more expressive architectures such as Graph Attention Networks (GAT), Graph Transformers Rampášek et al. (2022), and subgraph-aware models Bouritsas et al. (2020). These models have shown improved performance in tasks requiring long-range dependency modeling or structural pattern recognition, which could be especially beneficial for capturing both global chemical context and local substructural motifs in molecular graphs. Additionally, pretraining strategies for GNN models Hu et al. (2020b) represent an active area of research. Leveraging unsupervised or self-supervised pretraining on large chemical databases could improve generalization. On kernel methods, our results showed that kernels using very different information -WL vs. Shortest Path - had very similar performance. This points us towards exploring ways of aggregating information from different kernels. Moreover, while trainable encoders (to our knowledge) would not work in a kernel setting, it would be interesting to find a way to embed the categorical attributes into a real valued space so we could test kernels that are designed for continuous valued attributes, therefore leveraging all the information available.

-
10. The ogbg-molhiv dataset requires specific versions of PyTorch, while GraKeL has its own version dependencies, creating considerable friction in the data conversion process. These library version incompatibilities necessitated complex workarounds and limited the seamless integration of different graph learning approaches.
 11. Choosing the right combination of layers, aggregation functions, and pooling strategies often came down to trial-and-error or replicating choices from prior work, rather than principled model selection.
 12. Graph Transformers, deep GNN architectures with virtual nodes or advanced pooling mechanisms.
 13. Similar computational demands arose with kernel methods, while WL kernels are designed to be computationally efficient in training, large sets of graphs still require large RAM capabilities to store the Gram matrix.

Appendix A. Molecular Graph Features

A.1 Node Features

Each atom (node) in the molecular graph is described by a 9-dimensional feature vector capturing its chemical properties. Table 3 summarizes these features.

Table 3: Atomic-level (node) features used in the molecular graphs.

Feature	Description
Atomic Number	Number of protons; identifies element
Chirality	3D orientation around stereocenter
Degree	Number of covalent bonds
Formal Charge	Net electrical charge
Number of Hydrogens	Attached hydrogens (explicit/implicit)
Radical Electrons	Number of unpaired electrons
Hybridization	Orbital type (sp , sp^2 , sp^3)
Is Aromatic	Boolean; participates in aromatic system
Is in Ring	Boolean; part of cyclic structure

A.2 Edge Features

Each chemical bond (edge) is represented with a 3-dimensional feature vector. The bond features are outlined in Table 4.

Table 4: Bond-level (edge) features used in the molecular graphs.

Feature	Description
Bond Type	Bond order (single, double, triple, aromatic)
Bond Stereochemistry	Spatial configuration (cis, trans, unspecified)
Is Conjugated	Boolean; part of conjugated system

A.3 Graph statistics and Class imbalance

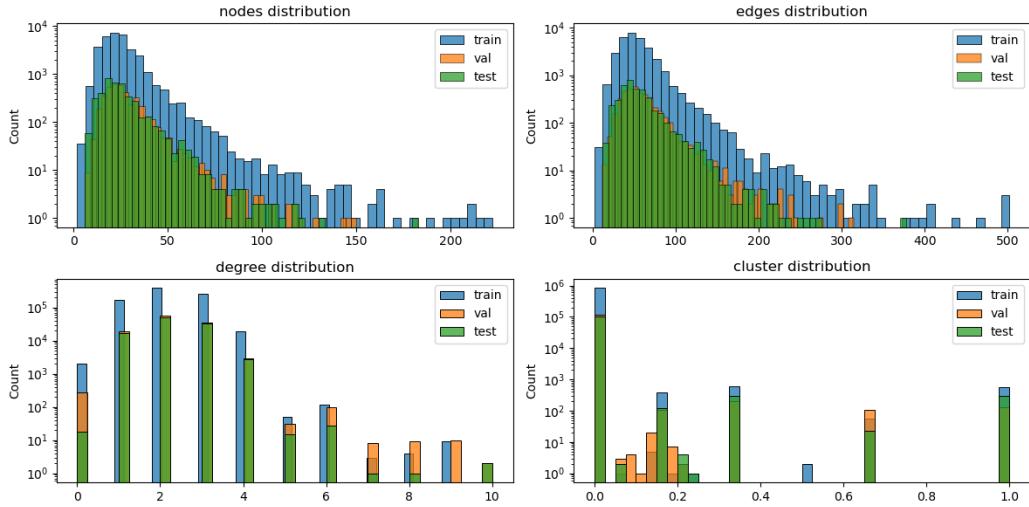


Figure 1: Distributions of graph statistics (nodes, edges, degree, clustering) across the training, validation, and test splits of the OGBG-MolHIV dataset.

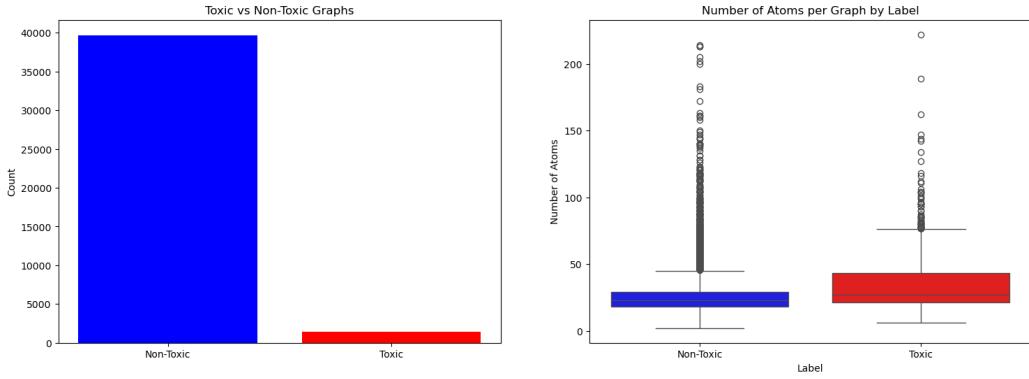


Figure 2: Class distribution and molecule size distribution by label

Appendix B. GNN result tables

Normalization	GCN	Gated GCN	Virtual Node GCN
BatchNorm	0.745, 0.706	0.767, 0.723	0.671, 0.649
LayerNorm	0.744, 0.672	0.733, 0.718	0.654, 0.620
InstanceNorm	0.753, 0.719	0.673, 0.707	0.652, 0.626
GraphNorm	0.710, 0.649	0.744, 0.686	0.639, 0.620

Table 5: Validation and Test ROC-AUC for Different Normalizations (Mean Pooling).

Pooling method	GCN	GCN + VN	Gated GCN	GIN	GIN + VN
Mean	0.807, 0.752	0.820, 0.763	0.802, 0.773	0.825, 0.759	0.809, 0.785
Sum	0.757, 0.762	0.770, 0.700	0.768, 0.739	0.837, 0.781	0.768, 0.743
Max	0.677, 0.690	0.574, 0.581	0.734, 0.725	0.821, 0.742	0.671, 0.620
Attention	0.798, 0.804	0.790, 0.746	0.795, 0.765	0.809, 0.712	0.810, 0.753
Set2Set	0.744, 0.697	0.779, 0.763	0.745, 0.723	0.797, 0.759	0.814, 0.769

Table 6: Validation and Test ROC-AUC for different pooling strategies (All with Batch Normalization). We consider the type of pooling a hyperparameter, that is why 0.804 is not reported as best test result.

Appendix C. On the Expressiveness of Pooling Operators

Let $\mathcal{X} = \{x_1, \dots, x_n\}$ be a multiset of node embeddings. Common pooling operators include:

$$\text{Sum: } h(\mathcal{X}) = \sum_{x \in \mathcal{X}} f(x), \quad \text{Mean: } h(\mathcal{X}) = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} f(x), \quad \text{Max: } h(\mathcal{X}) = \max_{x \in \mathcal{X}} f(x)$$

While all three are permutation invariant, only the sum aggregator is injective over bounded multisets, making it capable of uniquely identifying neighborhoods with differing multiplicities. For example, $\mathcal{X}_1 = \{f(a), f(a)\}$ and $\mathcal{X}_2 = \{f(a), f(a), f(a)\}$ yield different outputs under sum, but identical ones under mean and max.

Mean pooling discards cardinality and preserves only distributional information:

$$h(\mathcal{X}_1) = \frac{1}{2}(f(r) + f(g)), \quad h(\mathcal{X}_2) = \frac{1}{3}(f(r) + 2f(g))$$

which may coincide under certain feature symmetries. Max pooling is more restrictive, treating multisets as sets by returning the dominant feature, thus ignoring both frequency and distribution.

Attention-based pooling generalizes mean by learning a weight $\alpha(x)$ for each node:

$$h(\mathcal{X}) = \sum_{x \in \mathcal{X}} \alpha(x) f(x)$$

offering adaptivity but lacking injectivity. Set2Set introduces iterative attention to refine this process, increasing expressiveness at the cost of computational complexity.

In summary, sum pooling is strictly more expressive than mean and max. Attention and Set2Set provide flexible alternatives, but cannot recover full multiset information without additional assumptions.

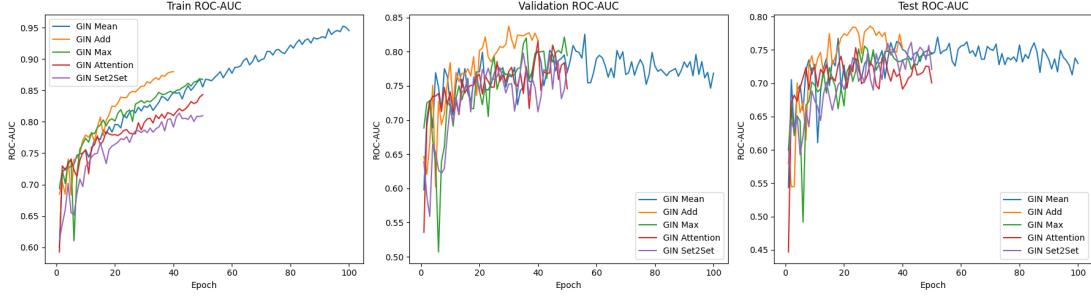


Figure 3: GIN: Pooling Methods ROC-AUC Comparison

Appendix D. Time Complexity for Graph Networks

Model	Pooling Method	Pooling Complexity	Total Complexity
GCN	Mean / Sum / Max	$O(n)$	$O(nd^2L)$
GCN	Attention	$O(nd)$	$O(nd^2L + nd)$
GCN	Set2Set	$O(nTd)$	$O(nd^2L + nTd)$
GIN	Mean / Sum / Max	$O(n)$	$O(nd^2L)$
GIN	Attention	$O(nd)$	$O(nd^2L + nd)$
GIN	Set2Set	$O(nTd)$	$O(nd^2L + nTd)$

Table 7: Pooling and total complexity for GCN and GIN variants. Message passing complexity is $O(nd^2L)$ for all models, where n is the number of nodes, d the feature dimension, L the number of layers, and T the number of iterations for Set2Set.

References

- Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, Leslie O’Bray, and Bastian Rieck. Graph kernels: State-of-the-art and future challenges. 2020.
- Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL <https://arxiv.org/abs/2006.09252>.
- Xavier Bresson and Thomas Laurent. Residual gated graph convnets. In *ICLR Workshop*, 2017. URL <https://arxiv.org/abs/1711.07553>.
- Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2021. URL <https://arxiv.org/abs/2009.03294>.
- C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.

- 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. PMLR, 2017.
- William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.
- 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. *arXiv preprint arXiv:2005.00687*, 2020a.
- Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020b.
- Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.
- Greg Landrum. Rdkit: Open-source cheminformatics, 2006. <http://www.rdkit.org>.
- Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *ICML*, 2019.
- Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *ICLR*, 2016.
- Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman Publishers, San Mateo, CA, 1988.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- John C Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in large margin classifiers*, pages 61–74. MIT Press, 1999.
- Lukas Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, and Guy Wolf. Recipe for a general, powerful, scalable graph transformer. In *International Conference on Learning Representations (ICLR)*, 2022. URL <https://openreview.net/forum?id=eSHj9D0ZtD>.
- Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *Journal of Machine Learning Research*, 21(54):1–5, 2020.
- Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.
- Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Cody Geniesse, Ajay S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

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

Xinwei Zhang, Yifan Feng, and Yue Gao. HyperFusion: Multi-model ensemble on hypergraph. <https://github.com/zhangxwww/HyperFusion>, 2023. Accessed: 2025-05-06.