Generalization and Expressivity of Graph Neural Networks in Practice

Sebastian Lund

June 9, 2023

Summary

Graph neural networks (GNNs) have been widely used for various tasks on graph-structured data, such as node classification, link prediction, and graph classification. Graphs are very different from structured data such as images, as there is not consistent structure to rely on. Every node can have an arbitrary number of neighbors and it must somehow aggregate information from them. GNNs are very general in that you can convert any structured data such as images into graphs by throwing away the structural information. Noone would seriously consider this as most methods on structured data heavily rely on the structure for invariance and equivariance in the architecfture. The fact that GNNs are so general also means that they are hard to develop. The arguable most popular type of GNN are message-passing neural networks (MPNNs). MPNN only interact through local interactions, which gives them a low computational blueprint. They unfortunately also come with the downside that they are limited in expressivity. Much work has gone into studying and improving the theoretical expressivity of these models in an effort to design more general GNNs. These results often come in the form of aligning the expressive power with the Weisfeiler-Lehman (WL) test (Weisfeiler and Leman, 1968), which is a graph isomorphism test. It is very useful to consider expressivity with respect to isomorphism. If two (sub-)graphs are isomorphic to a GNN, it cannot assign them different labels. This fact has brought the design of so-called universal MPNNs. These are MPNNs that, theoretically, are as strong as the WL-test itself. Unfortunately these theoretical results have strong practical limitations. Throughout this paper we evaluate a set of theoretical results. We show that universal MPNNs cannot learn injective functions, by showing that they cannot emulate steps of the WL-test. We go on to show that universal MPNNs can in fact learn to emulate less powerful MPNNs on bounded size graphs, confirming a theoretical claim. To evaluate whether MPNNs generalize beyond the bounded size graphs, we construct 3 simple logical node classifiers and show that MPNNs generally do not learn to generalize far outside of the training distribution even for simple problems with exact solutions. The experiment also shows that less powerful MPNNs perform and generalize much better than universal MPNNs on some tasks. This motivates us to test whether less powerful MPNNs can gain expressive power by being graph encoders in GNNs that

add additional structural information. We use a framework called **ESAN** (Bevilacqua et al., 2022) that adds structural information by splitting a graph into a bag of subgraphs to where the MPNN is applied. We find that **ESAN** versions with weaker MPNNs perform on par with universal MPNNs for two real world datasets of molecules. In this process we even find that an MPNN with mean aggregation (not universal) outperforms universal MPNNs as encoders. Based on this result we go on recommend that new structural information frameworks should consider evaluating on an additional set of weaker MPNNs in their experiments.

Contents

1 Introduction **3**

2 Related Work **4**

3 Graph Neural Networks **4**
3.1 Notation 4
3.2 Message-Passing Neural Networks 5
3.3 Subgraph Graph Neural Networks 7

4 Expressive Power of GNNs **8**
4.1 Graph Isomorphism 8
4.2 First Order Logic 11
4.3 Graph Biconnectivity 12
4.4 Known Expressivity Results 12

5 Expressive Power in Practice **14**
5.1 Emulating The WL Test 15
5.2 Emulating "Weaker" GNNs 17

6 Logical Expressiveness in Practice **19**
6.1 Discussion 20

7 Higher Order GNNs With Weaker Encoders **21**
7.1 Discussion 24

8 Conclusion **24**

A Experimental Setup: Emulating The WL Test **28**

B Experimental Setup: Emulating "Weaker" GNNs **28**

C Experimental Setup: Logical Node Classifiers **29**

D Experimental Setup: Higher Order GNNs With Weaker Encoders **29**

1 Introduction

Graph Neural Networks have in recent years become a dominant tool for learning on unstructured data in the form of graphs. They are used for both node classification, graph classification, link prediction and graph representation learning. The most common class of GNN are Message-passing neural networks (MPNNs), GNNs that only have local interactions in a graph. This is mainly because local interactions have a low computational burden. Unfortunately, theoretical results put hard limits on the power of MPNNs. Much work has gone into aligning the power of GNNs with the Weisfehler-Lehman (WL) test (Weisfeiler and Leman, 1968). This has resulted in the advent of so-called maximally expressive MPNNs like **GIN** (Xu et al., 2019), that supposedly are as expressive as an MPNN can be due to their aggregation function. The interplay between the WL test and GNNs has also lead to the development of GNNs based on the test such as Higher Order GNNs(Morris et al., 2019). While higher order GNNs are indeed expressive they are much more computationally expensive than MPNNs, which has lead to a lack of adoption on real world problems. To bridge the computational gap, many recent GNNs augment MPNNs in ways that makes them more expressive. Such as adding random node features (Sato et al., 2021) or substructure information (Bouritsas et al., 2022). While these works could employ any MPNN, they often opt for *universal* MPNN as their graph encoders. While this may seem like a good idea based on the theoretical results, we will show in this paper that it is sometimes worth considering less powerful MPNNs.

The main contributions on this paper are as follows:

- We show that universal MPNNs cannot learn injective functions in practice. They cannot emulate the first few steps of the WL test. (Section 5.1)
- We show that universal MPNNs can learn to emulate less powerful aggregation functions in practice. We find evidence for the hypothesis that **GIN** has trouble learning to disentangle its own representation from that of its neighborhood. (Section 5.2)
- We show that MPNNs do not generalize to graph size in practice on 3 distinct logical node classifiers. Results show that less powerful MPNNs are more suited for generalization in specific tasks. (Section 6)
- We empirically validate that all MPNNs are able to leverage structural information to increase their expressivity in real world problems. (Section 7)
- We empirically validate that non-universal MPNNs with structural information can outperform universal MPNNs with structural information in real world problems. (Section 7)

2 Related Work

This section is mainly our previous work in Lund (2023). We cover relevant related work throughout the paper.

GNN Expressivity Expressivity of GNNs has been studied theoretically, drawing equivalences to methods such as the Weisfeiler-Lehman (WL) test from Weisfeiler and Leman (1968). It has been shown that message passing GNNs are at most as powerful as the WL test (Morris et al., 2019; Xu et al., 2019). Xu et al. (2019) highlight that the aggregation scheme used must be injective and shows a way to construct arbitrary aggregators from a sum as an extension of the result from sets in Zaheer et al. (2017) to multisets. Wagstaff et al. (2019) show that the construction requires highly discontinuous mappings which are impractical to learn by neural networks. Various other works describe alternative ways to learn aggregation functions (Corso et al., 2020; Pellegrini et al., 2021). The study of the relation between WL and GNNs has lead to designs of higher order GNNs (Morris et al., 2019; Maron et al., 2019). The limitations of the WL test have lead to designs using higher-order graph structures from topology (Bodnar et al., 2021b,a). And exploit the ability to break large graphs into substructures (Bevilacqua et al., 2022). In this work we consider the proposition of using less powerful message passing GNNs along with substructure information. Barceló et al. (2020) draw an equivalence between the expressivity of logic classifiers in graded modal logic and that of message passing GNNs. They further show sufficient conditions to express FOLC_2 classifiers, while in practice the learned GNNs generalize poorly to even slightly larger graphs. In this work we design even simpler logic classifiers and show that they do not generalize in a more thorough experiment for a much larger set of GNN architectures.

3 Graph Neural Networks

In this section we will define the Graph Neural Networks (GNNs) that we will consider throughout the paper. We mainly focus on the widely used Message-Passing Neural Networks (MPNNs) and describe the architectures used in Section 3.2. In Section 3.3 we describe a recent method for improving the expressivity of MPNNs by applying simple GNNs to a set of subgraphs and aggregating.

3.1 Notation

In this section we define the notation that we will use throughout the paper for graphs and related concepts. We use $\{\}$ to denote sets and $\{\!|\}$ to denote multisets. In this paper we only work with undirected graphs $G = (\mathcal{V}, \mathcal{E})$ with no self-loops, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Because edges are undirected we can represent them with sets $\{u, v\} \in \mathcal{E}$. For each node $v \in \mathcal{V}$ have $\mathbf{x}_v \in \mathbf{X}$ be the initial attributes/features of the node. We will in general only use categorical node features throughout the paper. We denote the multiset

of neighbors of a node $v \in \mathcal{V}$, the neighborhood, by $\mathcal{N}_v = \{\!\{ u \in \mathcal{V} : \{v, u\} \in \mathcal{E} \}\!\}$ and its degree as $D(v) = |\mathcal{N}_v|$. We will use the term *configuration* for a pair consisting of the features of a node, and its neighborhood $\left\langle \mathbf{x}_v, \{\!\{ \mathbf{x}_u : u \in \mathcal{N}_v \}\!\} \right\rangle$.

3.2 Message-Passing Neural Networks

The introduction to this section is based on our previous work in Lund (2023). Message-passing neural networks (**MPNN**s) (Gilmer et al., 2017) are a class of graph neural networks based on the idea that nodes communicate with their immediate neighbors through *messages* to update their representations. This can be seen as a *propagation* or *diffusion* of node representations.

More formally, the representation of a node u ($\mathbf{h}_u$) is a function of its features ($\mathbf{x}_u$) and the features of its multiset of neighbors $\{\!\{ \mathbf{x}_v \in \mathbf{X} : v \in \mathcal{N}_n \}\!\}$. GNNs usually consist of multiple consecutive layers to propagate information from further away. We denote the representation of node u at layer i by $\mathbf{h}_u^{(i)}$ and set $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.

One of the first and simplest **MPNN**s are the Graph Convolutional Network (**GCN**) (Welling and Kipf, 2016; Defferrard et al., 2016). In **GCN**s edges are weighed nonparametrically with a single scalar weight $c_{uv} \in [0, 1]$ for an edge between nodes u and v. This weight is usually based on the structure of the graph e.g. the degree of the nodes or, in directed graphs, the in-/out-degree. The representation of a node u in a **GCN** is then given by:

$$\mathbf{GCN}\mathbf{h}_u^{(i+1)} = \phi\Big(\sum_{v \in \mathcal{N}_u \cup \{u\}} c_{uv} \mathbf{h}_v^{(i)} \Big) \tag{1}$$

Where $\phi(x) = \sigma(W_\phi x)$, σ is a non-linearity and W_ϕ is a weight matrix. c_{uv} are usually restricted such that they sum to 1, making the new node representation a weighted average of the (transformed) neighborhood and itself. Notice how a **GCN** in some cases cannot distinguish its own previous features from that of its neighbor, see Example 3.1 below.

Example 3.1. Consider the assignment of $c_{uv} = (D(u)+1)^{-1}$ in a **GCN**, such that the aggregation becomes an unweighted mean. Now the following configuration of a node with one `Blue` neighbor ($\texttt{Red}, \{\!\{ \texttt{Blue} \}\!\}$) is indistinguishable from a configuration with one `Red` neighbor, namely ($\texttt{Blue}, \{\!\{ \texttt{Red} \}\!\}$). Both means are equal to $1/2\ \texttt{Blue} + 1/2\ \texttt{Red}$.
□

While invariance as to whether a node is a neighbor or not may be a useful inductive bias in some problems, it is not useful for all problems.

GraphSAGE (Hamilton et al., 2017) alleviates this problem by having ϕ operate on a concatenation of the node's previous layer representation and the aggregation. While Hamilton et al. (2017) proposes multiple aggregation functions (mean, lstm, max), we will only consider the commonly used max aggregation in this paper. The representation for **SAGE** is then given by:

$$\mathbf{SAGE}\mathbf{h}_u^{(i+1)} = \phi\Big(\mathbf{h}_u^{(i)} \parallel \max_{v \in \mathcal{N}_u} \psi(\mathbf{h}_v^{(i)}) \Big) \tag{2}$$

Where $\|$ is concatenation and ϕ and ψ are linear transformations followed by a non-linearity.

GCNs have also moved development in another direction. With the motivation of being able to assign different importance to nodes of the same neighborhood, Graph Attention Networks (**GAT**) (Veličković et al., 2018) were proposed. It can be thought of as making the c_{uv} weights from **GCN**s parametric and dependent on node features instead of degree. The neighborhood weights are now decided by a function $a(\mathbf{h}_u^{(i)}, \mathbf{h}_v^{(i)})$. It learns to compare the node features to weigh how important $\mathbf{h}_v^{(i)}$ is for the representation of node u ($\mathbf{h}_u^{(i+1)}$). $a(\mathbf{h}_u^{(i)}, \mathbf{h}_v^{(i)})$ is often softmax-normalized, such that the weights for the neighborhood sum to 1. The representation of a node u in a **GAT** is given by:

$$_{\mathbf{GAT}}\mathbf{h}_u^{(i+1)} = \phi\Big(\sum_{v \in \mathcal{N}_u} a(\mathbf{h}_u^{(i)}, \mathbf{h}_v^{(i)})\psi\big(\mathbf{h}_v^{(i)}\big)\Big) \tag{3}$$

Typically, ϕ, ψ and a are linear transformations followed by non-linearities or multi-layer perceptrons (MLPs).

The previous three GNNs all have normalizations ensuring that the aggregation is the same magnitude as the node representations. It has been shown empirically that this is not always desirable. One such GNN with empirical success are *Graph Isomorphism Network* (**GIN**) (Xu et al., 2019). **GIN** use *sum-decomposition* (Wagstaff et al., 2019) e.g. transformations before and after aggregation by sum. The representation of a node in **GIN** are given by:

$$_{\mathbf{GIN}}\mathbf{h}_u^{(i+1)} = \mathbf{MLP}\Big((1 + \epsilon^{(i)}) \cdot \mathbf{h}_u^{(i)} + \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(i)}\Big) \tag{4}$$

Where epsilon is a learned parameter for the weight of the nodes own previous representation.

Finally we consider the class of GNNs termed *Aggregate Combine GNNs* in Barceló et al. (2020). We consider two variants, one with mean aggregation (**AC-**μ) and one with sum aggregation (**AC-**Σ). The representation of a node in an **AC-**Σ and **AC-**μ are given by:

$$_{\mathbf{AC\text{-}}\Sigma}\mathbf{h}_u^{(i+1)} = \mathbf{MLP}\Big(\mathbf{h}_u^{(i)} \parallel \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(i)}\Big) \tag{5}$$

$$_{\mathbf{AC\text{-}}\mu}\mathbf{h}_u^{(i+1)} = \mathbf{MLP}\Big(\mathbf{h}_u^{(i)} \parallel \frac{1}{D(u)} \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(i)}\Big) \tag{6}$$

We have now described a set of distinct GNN architectures in the message passing regime. These GNNs all aggregate messages differently, and unexpectedly they have different expressivities. In Section 4.4 we cover known theoretical results for how expressive these GNNs are in comparison to one another.

In the upcoming section we give an overview of a conceptually simple GNN technique for improving the expressive power of MPNNs by applying them to multiple subgraphs individually.

3.3 Subgraph Graph Neural Networks

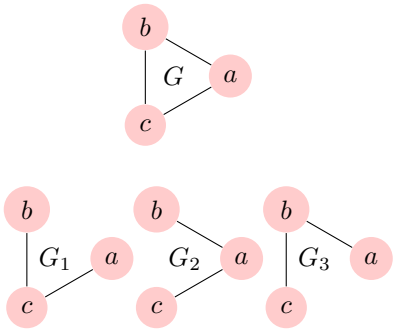
In the previous section we gave an overview of message-passing neural networks, describing how they communicate solely through local interactions along the edges in a graph. In many real world problems, especially molecular tasks, it can be necessary to consider higher-order graph structures such as triangles and rings. Unfortunately, MPNNs cannot count substructures consisting of 3 or more nodes (Chen et al., 2020), some intuition as to why is given in Section 4.1. Higher-order Graph Neural Networks (Morris et al., 2019) which are able to count substructure of size 3, are prohibitively computationally expensive. To combat this some works try to encode the appearance of certain substructures (Bouritsas et al., 2022), but deciding on the substructures requires domain knowledge. We use a more conceptually simple method of improving the expressivity of GNNs that does not rely on domain knowledge, Equivariant Subgraph Aggregation Networks (**ESAN**) (Bevilacqua et al., 2022). **ESAN** break up higher-order graph structures and symmetries by breaking graphs down into subgraphs. The idea is simple: Break a graph up into a bag of subgraphs and apply GNNs to each subgraph individually, aggregating at the end.

In **ESAN** the bag of subgraphs $S_G = \{\!\{ G_1, \ldots, G_m \}\!\}$ is generated from the orginal graph G by a subgraph selection policy π. Bevilacqua et al. (2022) only consider selection policies that select *spanning* subgraphs, that is subgraphs where only edges are removed, because it is computationally beneficial that all adjacency matrices are the same size. They describe 4 subgraph selection policies; node-deleted subgraphs (ND), edge-deleted subgraphs (ED), and ego-networks (EGO, EGO+). The node-deleted policy generates $m = |\mathcal{V}|$ subgraphs; for every node it generates a subgraph where that node has its edges removed. The edge-deleted policy generates $m = |\mathcal{E}|$ subgraphs; for every edge it generates a subgraph where that edge is removed. Figure 1 shows examples of the edge-deleted and node-deleted policies. The ego-networks policy generates $m = |\mathcal{V}|$ subgraphs; for every node it generates the subgraph induced by the nodes that are reachable in k hops. The EGO+ policy generates the same ego-networks, but marks the central node with a special feature. Recently it has been shown that aggregating over graphs with *marked* nodes implicitly encodes distance and biconnectivity information (Zhang et al., 2023).

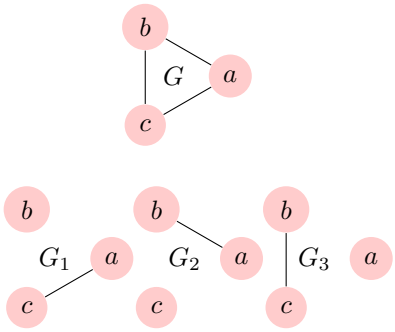
ESAN define two different architectures depending on whether the subgraphs should share information among themselves or act independently until the final readout. We will use the independent version in this paper, refer to Bevilacqua et al. (2022) for a detailed description of the information sharing process. When a subgraph selection policy is chosen, the function that acts on bags of subgraphs is comprised of three components:

$$F_{DSGNN} = E_{sets} \circ R_{subgraphs} \circ E_{subgraphs} \tag{7}$$

The subgraph encoder $E_{subgraphs}$; applying a GNN to each subgraph. The subgraph readout $R_{subgraphs}$ a graph level readout applied to each subgraph, e.g. a mean, max or sum. The set encoder E_{sets} to pool the set of subgraph readouts and classify the graph. We use DeepSets Zaheer et al. (2017) as the



(a) A traingle graph G and its edge-deleted subgraphs (ED)



(b) A traingle graph G and its edge-deleted subgraphs (ED)

Figure 1: Subgraph selection policies in the **ESAN** framework

set encoder E_{sets} and will vary $E_{subgraphs}$ by using the GNNs described in the previous section on real world datasets in Section 7.

To understand why we would want to improve the expressive power of MPNNs, we need to understand what they can and cannot represent. In the following section we will cover relevant theoretical results on the expressive power of GNNs.

4 Expressive Power of GNNs

Theoretical results are particularly useful for understanding the upper bounds of what GNNs are capable of. Much work has gone into bounding the expressive power of GNNs by a graph isomorphism test called the Weisfeiler-Lehman test. In this section we will describe why it can be useful to think in terms of graph isomorphisms. We will go on to describe useful theoretical results and throughout the paper test how well they hold up in practice.

4.1 Graph Isomorphism

This section is based on our previous work in Lund (2023). If two graphs with different labels are indistinguishable from each other it is impossible to classify them both correctly. It is therefore useful to understand which graphs can be distinguished by a specific GNN when studying its expressivity. In graph-theoretic terms, two graphs are *isomorphic* when they are indistinguishable. An *isomorphism* of graphs G and H is a bijective map f from the vertices of G to the vertices of H that preserves the edge relation. $f : \mathcal{V}(G) \to \mathcal{V}(H)$ where u and v are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. Figure 2 shows an isomorphism between two graphs preserving the edge relation. Specifically, the green node in both graphs has a single blue and yellow neighbor, and this pattern holds true for all other nodes.

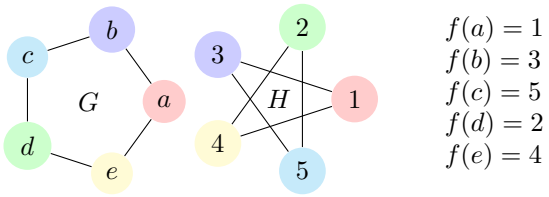


Figure 2: Two graphs G and H with their isomorphism f. Nodes are colored based on the bijection.

In the GNN setting we usually operate on graphs with node features $\mathbf{x}_u \in \mathbf{X}$ assigned to every node u. We also include this in the isomorphism, such that $\forall u \in \mathcal{V}(G), \mathbf{X}(u) = \mathbf{X}\big(f(u)\big)$. Figure 3 shows two graphs that have no isomorphism now that features are involved: In graph G every **Hot** node has a **Hot** neighbor, while in graph H all **Hot** nodes only have **Cold** neighbors, hence the edge relation and features cannot both be preserved by a bijective map.

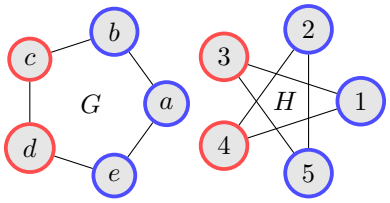


Figure 3: Two graphs G and H with binary node features $\{\mathbf{Hot}, \mathbf{Cold}\}$ shown with red and blue outlines, respectively. No feature preserving isomorphism exists.

There are currently no known polynomial time algorithms for determining whether two graphs are isomorphic in general (Schöning, 1988). However, there are efficient algorithms that don't fully characterize graphs up to isomorphism, but still distinguishing a significant number of nonisomorphic graphs. The Weisfeiler-Leman test is one such algorithm.

Weisfeiler-Leman Test The Weisfeiler-Leman algorithm is an isomorphism test based on color refinement (Weisfeiler and Leman, 1968). Inspired by Cai et al. (1992) we define the 1 dim WL (vertex refinement): For a graph G, given node features $\mathbf{x}_u \in \mathbf{X}$ for $u \in \mathcal{V}(G)$, let $W^0 : \mathcal{V}(G) \to \mathcal{C}$ be given by $W^0(v) = \text{hash}(\mathbf{x}_v)\ \forall v \in \mathcal{V}(G)$ where $\mathcal{C}$ is a set of *colors* and hash is injective. We then define W^{r+1} from W^r by assigning a new color to each node based on their neighborhoods:

$$W^{r+1}(v) = \text{hash}(W^r(v), \{\!|\ W^r(u) | u \in \mathcal{N}_v\ |\!\})\ \forall v \in \mathcal{V}(G) \tag{8}$$

We keep refining the coloring until at some level the refinement stabilizes such that $W^{r+1} = W^r$, as in Cai et al. (1992) we will use $\overline{W}$ to denote the stable

refinement of W^0 and denote the multiset of colors in the stable refinement for graph G by $\mathrm{WL}(G) = \{\!\{ \overline{W}_G(v) | v \in \mathcal{V}(G) \}\!\}$. Two graphs G and H are not isomorphic if $\mathrm{WL}(G) \neq \mathrm{WL}(H)$. If $\mathrm{WL}(G) = \mathrm{WL}(H)$ the graphs may or may not be isomorphic.

The 1-dimensional WL test fails to distinguish all regular graphs with n nodes and degree d when all node features are the same. It does however correctly identify that G and H in Figure 3 are not isomorphic, a **Hot** node with a **Hot** neighbor (c or d in G) is assigned a unique hash (color) which cannot occur in H. This highlights an important point, if WL on two graphs disagree at any step, they cannot be isomorphic because the coloring is injective so the test can stop early. Figure 4 shows a failure of the WL test on two non-isomorphic graphs with different substructures.

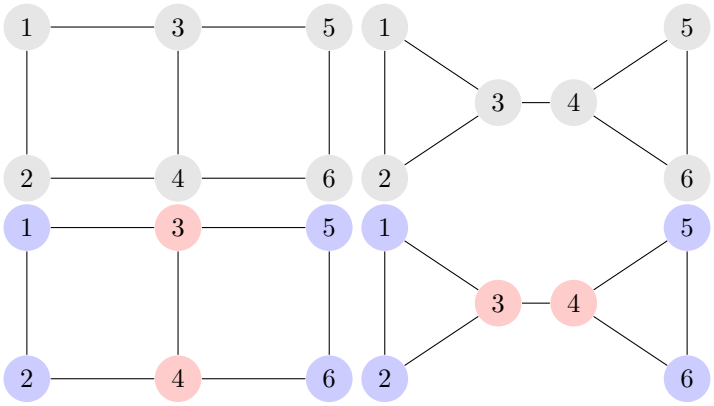


Figure 4: 1-WL converges after one step and fails to distinguish the two graphs. It cannot count substructures: The right graph contains two triangles but the left does not so the two are not isomorphic.

As briefly mentioned earlier, the WL test can be extended to higher dimensions to be more powerful. This is described in detail in Weisfeiler and Leman (1968) and an altered version in Cai et al. (1992).[1] We will keep it brief as we only consider **MPNN**s which are at most as expressive as vertex refinement. The main result of Cai et al. (1992) is to show that k dimensional WL (k-WL) has the same stable refinement for two graphs if and only if the two graphs satisfy the exact same formulae in k variable *first order logic with counting* (FOC_k). The proof relies on mapping both k-WL and FOC_k to a combinatorial game of placing k pebbles and showing a winning strategy exists for the defending player. The vertex refinement WL described above happens to be equal to 2-WL, so it cannot distinguish graphs which satisfy the same FOC_2 formulae. Knowing

[1]The discriminating power of k-WL from Weisfeiler and Leman (1968) is equivalent to $(k-1)$-WL from Cai et al. (1992) for $k \geq 3$ (Grohe and Otto, 2015).

that, Figure 4 makes total sense: We cannot count the number of triangles with less than 3 variables! Running 3-WL would show that the two graphs are in fact not isomorphic. This connection to logic formula and the versatility of first order logical grammar is a strong basis for evaluating GNNs on logic formula.

4.2 First Order Logic

The equivalence between the WL test distinguishing graphs and first order logic with counting gives us a new perspective on the failure modes of the WL test, which in turn are the failure modes of MPNNs. In this section we describe first order logic, and how it related to FOC_k. We consider First order predicate logic (FO) where each vertex has a unique color, such that we can distinguish them. Below is an example of a node classifier ϕ_1:

$$\phi_1(x) := \text{Blue}(x) \wedge \exists y(E(x,y) \wedge \text{Green}(y)) \tag{9}$$

It states that $\phi(x)$ is true if x is Blue and there exists another node y, which is a neighbor to x and is Green. To motivate the need for counting, lets say we want to ask if x has two green neighbors. The logical node classifier would be defined by:

$$\phi_2(x) := \text{Blue}(x) \wedge \exists y \exists z(E(x,y) \wedge \text{Green}(y) \wedge E(x,z) \wedge \text{Green} \wedge x \neq y) \tag{10}$$

What first order logic with counting allows us to do is write the following instead

$$\phi_3(x) := \text{Blue}(x) \wedge \exists^{=2} y(E(x,y) \wedge \text{Green}(y)) \tag{11}$$

Now the uniqueness of each y is implicit instead of explicitly stating that $y_1 \neq y_2$ etc. and it becomes much less verbose. ϕ_3 above is an FOC_2 formula because we use counting quantifiers and there are two free variables x and y. But, just because we have k counting quantifiers in our node classifier does not mean that we are necessarily FOC_{k+1}. If we can write a formula and reuse variables that is allowed:

$$\phi_4(x) := \text{Blue}(x) \wedge \exists^{=2} y(E(x,y) \wedge \text{Green}(y)) \wedge \exists^{=2} y(E(x,y) \wedge \text{Blue}(y)) \tag{12}$$

ϕ_4 is still in FOC_2. An example of an FOC_2 formula is counting the number of triangles in the graph. An FOC_3 counting the number of triangles touching x looks like

$$\phi_5(x) := \exists^{=2} y \exists z(E(x,y) \wedge E(y,z) \wedge E(z,x)) \tag{13}$$

The formula reads that x touches exactly one triangle and it is FOC_3, notice how it can distinguish the graphs in Figure 4, you can find a node that satisfies it in the right graph but not in the left. We will return to first order logic with counting in Section 6 as a useful tool for testing the ability of GNNs to generalize.

In the following section we describe a completely different way of looking at the expressivity that is related to the **ESAN** architecture.

4.3 Graph Biconnectivity

In a recent paper Zhang et al. (2023) describe a different approach to evaluating the expressivity of GNNs. They look at expressivity through the lens of *graph biconnectivity*. We briefly cover these results because they allude to the kinds of expressivity that are gained by using the **ESAN** framework. To understand biconnectivity we must first understand connectivity. A graph is connected if there is a path from each node to every other node. For example; considering the entirety of Figure 4 as one graph, it is unconnected. But the graph does have 4 connected components.

A *cut vertex* or an *articulation point* is a vertex, that if removed from the graph, increases the number of connected components. So node 3 and 4 in both right-side graphs of Figure 4 are **cut vertices**. Similarly **cut edges** increase the number of connected components, so the edges between 3 and 4 are cut vertices on the right graphs.

The idea of expressivity through the lens of biconnectivity, is whether the GNN can identify cut vertices and edges. **ESAN** was, prior to Zhang et al. (2023), the only non-WL-3 GNN able to detect cut vertices and edges. The reason that **ESAN** can detect these structures is because node markings (such as those in EGO+) implicitly encode distance information as per (Zhang et al., 2023, Lemma C.19).

Now that we have some understanding as to why **ESAN** could provide useful information for MPNNs. We will cover the main limitations and expressivity results of them in the next section.

4.4 Known Expressivity Results

The theoretical expressive power of some neural network architectures for graphs have been proven formally in terms of the WL-test. In this section we summarize the main results and their implications. We will start with the results for local interactions on graphs: Message-passing neural networks.

Xu et al. (2019) show that message-passing neural networks are at most as expressive as the WL-1 test in the following lemma:

Lemma 1. *(Xu et al., 2019, Lemma 2) Let G_1 and G_2 be any two non-isomorphic graphs. If a message-passing neural network $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ maps G_1 and G_2 to different embeddings, the Weisfeiler-Lehman graph isomorphism test also decides G_1 and G_2 are not isomorphic*

Recall how each step of the WL test ($W^i(v)$) injectively maps the features of a node (v) and its multiset of neighbors ($\mathcal{N}_v$) to a hash. To intuit Lemma 1, consider that each layer ($\phi^i(v)$) of message-passing neural network are functions with the exact same domain. At most the message-passing layer $\phi^i(v)$ can be injective, such that it maps G_1 and G_2 to different embeddings if and only if the WL test also does, as $W^i(v)$ is injective. If $\phi^i(v)$ is not injective it is strictly less powerful than the WL test as there are now more graphs G_1 and G_2 who must share embeddings.

The search for a maximally powerful message-passing neural network is therefore mainly the search for an injective function on multisets. Unfortunately, most common set aggregators are not injective on multisets. Example 4.1 below shows non-injective cases for *mean*, *max* and *attentional* aggregation.

Example 4.1. Consider the following multisets:

$$a : \{\!\{ \texttt{Blue}, \texttt{Red} \}\!\} \quad b : \{\!\{ 2 \circ \texttt{Blue}, 2 \circ \texttt{Red} \}\!\} \quad c : \{\!\{ \texttt{Blue}, 2 \circ \texttt{Red} \}\!\}$$

We treat the attributes as one-hot vectors. E.g. $\texttt{Blue} = \begin{bmatrix}1\\0\end{bmatrix}$ and $\texttt{Red} = \begin{bmatrix}0\\1\end{bmatrix}$. Only sum aggregation is injective on distributions:

$$\text{sum}(a) = \begin{bmatrix}1\\1\end{bmatrix} \neq \text{sum}(b) = \begin{bmatrix}2\\2\end{bmatrix}$$

$$^{2}\text{att}(a) = \text{mean}(a) = \max(a) = \text{att}(b) = \text{mean}(b) = \max(b) = \begin{bmatrix}1\\1\end{bmatrix}$$

Element wise max aggregation is only injective on sets:

$$\max(a) = \max(b) = \max(c) = \begin{bmatrix}1\\1\end{bmatrix}$$

□

While the example shows that distributional aggregation cannot count in general, they can use heuristics to approximate counting, this is one of the motivations for ideas such as random node features (Sato et al., 2021). Interestingly, when distributional aggregation are applied in the **ESAN** framework (Section 3.3) with node labelling on the central node of ego nets (EGO+) they can count when they neighbor the central node. The fact that only one node per graph can have the feature allows the aggregation to consistently get the reciprocal of the degree. We show in Section 7 that this leads to good results on real world datasets. To our knowledge it is currently an open question what the expressive power of distributional aggregation in **ESAN** is in relation to other GNNs.

In the quest for a universal aggregation function on multisets Xu et al. (2019) and Wagstaff et al. (2019) concurrently extend the work of Zaheer et al. (2017) on set aggregation to show that any multiset aggregation function can be constructed with a *sum-decomposition*, a sum aggregation and two combination functions ρ and ϕ, in the following lemma.

Lemma 2. *(Xu et al., 2019, Lemma 5) Assume $\mathcal{X}$ is countable. There exists a function $\rho : X \to \mathbb{R}^n$ so that $h(X) = \sum_{x \in X} \rho(x)$ is unique for each multiset $X \subset \mathcal{X}$ of bounded size. Moreover, any multiset function f can be sum-decomposed as $f(X) = \phi\big(\sum_{x \in X} \rho(x)\big)$ for some function ϕ*

[2]For this equality to hold we assume the attention to every element is a constant e.g. $a(\cdot, \cdot) = 0$. But att(a)=att(b) still holds in general for any $a(\cdot, \cdot)$.

Note that Lemma 2 does not tell us about the existence of a *continuous* sum-decomposition. Wagstaff et al. (2019) argue for the importance of this distinction, as the universal approximation theorem of neural network only applies to continuous functions (Cybenko, 1989). The fact that the function has to be continuous puts it under serious constraints for sum-decomposition, first the size of the multiset must be bounded, and secondly the dimension in which the sum is performed, has to be at least as high as the maximal number of elements. This is formalized in Theorem 1 from Wagstaff et al. (2019) below.

Theorem 1. *(Wagstaff et al., 2019, Theorem 4.4) Let $f : \mathbb{R}^{\leq M} \to \mathbb{R}$ be continuous. Then f is permutation-invariant if and only if it is continuously sum-decomposable via $\mathbb{R}^M$*

Under these constraints, given that 2 states $X \subset \mathcal{X}$ of bounded size, it is possible to construct continuous injective functions with sum-decomposition. Unfortunately, the functions are just entirely unmanageable in practice. It is an open question as to whether a practical solution exists, but we can provide impractical ones: One way is to map each unique $x \in X$ to a one-hot vector which can be done with a (very wide) MLP, doing the sum in the dimension of the one-hot vector and repeating at every layer, this quickly leads to an explosion in the size of the hidden dimension. Another way is to try to pack the unique x into single numbers with products of primes, but these numbers will be unfeasibly huge, and packing/unpacking them will again require huge dimensions.

This leaves us with a few questions for GNNs with sum-decomposition. Firstly, we would like to know how well they can learn injective functions in practice in settings with bounded graph sizes. We show in Section 5.1 that they do in fact struggle with learning injective functions. Secondly, the results for sum-decomposition apply to representing any aggregation function, not just injective ones. It is useful to know whether the limitations also impact the ability to learn simple aggregation functions like mean and max. We show in Section 5.2 that this mainly impacts **GIN** which heavily relies on sum-decomposition to disentangle its internal representation. Thirdly, the sum-decomposition is intrinsically tied to the maximal degree due to Theorem 1. It is important to know when, if ever, these GNNs can generalize beyond the training distribution. We show in Section 6 that they should not be expected to generalize even on simple logical classifiers.

5 Expressive Power in Practice

In this section we will try to systematically evaluate whether the theoretical expressivity results hold in practice when GNNs are learned with gradient descent. We test two claims in decreasing order of difficulty. In Subsection 5.1 we evaluate whether GNNs with sum-decomposition can emulate the WL Test, as described in the above section, this is theoretically possible as the maximal degree is bounded in the experiments. In Subsection 5.2 we evaluate whether

the theoretical expressivity ordering of GNNs holds in practice, by testing if GNNs can emulate random instantiations of "less powerful" GNNs.

5.1 Emulating The WL Test

In this section we will test how well, if at all, GNNs with sum-decomposition can learn injective functions in practice. Theoretically, if we have graphs with bounded degrees, there exists impractical instantiations of both GIN and AC-Σ that are injective for all node and neighborhood pairs, but can they be learnt?

In an attempt to evaluate this we will consider the task of emulating the WL-test. While the overall goal is to test whether GNNs can emulate the full WL-test, we would gain useful information from evaluating on gradually harder tasks. Fortunately, we can transform the test into gradually harder node classification tasks by using the node hashes at different *steps* of the WL-test as node labels. Recall how the first step of the WL-test just injectively maps each node and and their neighborhoods features to a hash e.g. $W^1(v) = \text{hash}(\mathbf{x}_v, \{\!| \mathbf{x}_u | u \in \mathcal{N}_v |\!\})$. We could then do multiclass classification with a class for each unique hash. In practice there are way too many unique hashes at the later WL steps to feasibly do multiclass classification, so we will instead transform the problem into binary node classification.

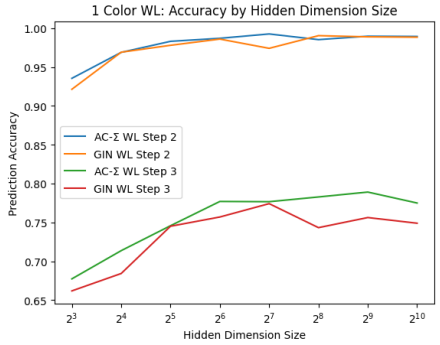
To generate binary labels from hashes we can define a set of relabelling functions $\rho_i : \mathcal{C} \to \{0, 1\}$ and assign the label of each node v to $\rho_i(W^k(v))$. To ensure that the labelling function is not biased we randomly fix 5 different relabelling functions $\rho_1, \ldots, \rho_5$ such that we can average the experimental results over them. In practice $\rho_i(\text{hash})$ is simply the i'th bit of the node hash.

An astute reader may be worried that the node classifiers, with the relabelling above, are no longer injective, which is true. But no real world classification task is injective either, what we are actually looking for is the ability to learn injective representations. The idea is that we are testing for any two nodes v, u along with their respective k-hop neighborhoods with different hashes ($W^k(v) \neq W^k(u)$), whether the GNN is able to map any such u and v to different representations before the classification head. This is tested because there is a non-zero probability that the labels do not match after the relabelling ($P(\mathbf{Y}_v \neq \mathbf{Y}_u) > 0$), resulting in an irreducible error if they are assigned to the same label.

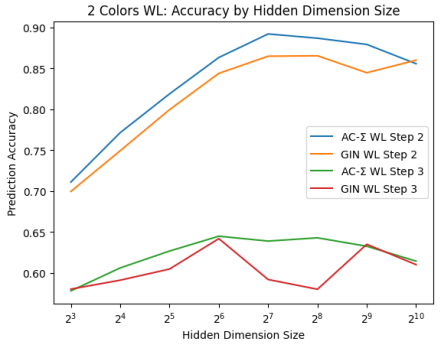
As mentioned earlier, there are exceedingly many unique hashes at the later steps of the WL-test, which is further exacerbated by having many distinct node features. To first test in a simple setting with less unique hashes we generate random Erdős–Rényi (Erdős et al., 1960) graphs where all nodes have the same initial features. We refer to this setting as ***1 Color***. To test in a more challenging setting we also generate random graphs with 2 distinct uniformly distributed initial node features. We refer to this setting as ***2 Colors***. We then train k-layer GNNs with a fixed hidden dimension on the k-th WL step.

	WL Step 1		WL Step 2		WL Step 3	
#Colors	***1Col***	***2Col***	***1Col***	***2Col***	***1Col***	***2Col***
#Unique	14	116	4k	9k	80k	350k
AC-Σ	100%	100%	98.7%	84.0%	77.7%	60.6%
GIN	100%	100%	98.6%	83.0%	75.7%	60.0%
GCN	58.6%	61.3%	59.6%	56.4%	56.3%	55.2%

Table 1: Number of node colors, number of unique hashes and mean binary prediction accuracy over 5 different binary relabelings at various steps of the WL-1 test for each GNN type (row). For brevity we use k for thousands e.g. $80k = 80.000$



(a) Prediction accuracy by hidden dimension size for 2nd and 3rd step of the WL test on ***1 Color***. The x-axis is log scale.



(b) Prediction accuracy by hidden dimension size for 2nd and 3rd step of the WL test on ***2 Colors***. The x-axis is log scale.

Figure 5

The mean accuracy over all relabelings for the first three steps of the WL-test on ***1 Color*** and ***2 Colors*** are reported in Table 1. The second row in the table shows the number of unique hashes in the training set, growing rapidly with both number of WL steps and the number of distinct features. Refer to Appendix A for the full experimental setup. Experiments in Table 1 are with a fixed hidden dimension size. To test whether the size of the hidden dimension matters we plot the accuracy for increasing the hidden dimension size from 2^3 to 2^{10} for ***1 Color*** and ***2 Colors*** in Figures 5a and 5b. Accuracy for 1st Step of WL is not shown in the figures as all dimensions achieve 100% accuracy in both tasks.

5.1.1 Discussion

As can be seen in Table 1, GNNs with sum-decomposition are able to perfectly emulate the first step of the WL-test in these simple settings. This is somewhat unsurprising, especially in the ***1 Color*** case, as it is essentially just counting

the number of neighbors. It is worth noting that GNNs with a hidden dimension as low as 8 achieve 100% accuracy at the first step, showing that a very low dimensional MLP can partition 116 distinct configurations into two uncorrelated classes. But, already at the second step, when the number of unique configurations are in the thousands, the accuracy drops. Figure 5 shows that this error is not due to lack of hidden dimension size, hinting at the idea that GNNs are incapable of fully learning injective functions. This is further highlighted by the accuracy at the third step, which is barely above random for ***2 Colors***.

In this section we have attempted to empirically validate the position of GNNs with sum-decomposition in the expressivity hierarchy, and we have found major limitations to how well they can learn injective functions. Theoretical results putting sum-decomposition above other aggregations in expressivity rely on assumptions of universality, which no longer hold without injectivity. We are now left with the question as to whether these GNNs are in fact universal. In the following section we will attempt to test whether GNNs with sum-decomposition can emulate less powerful GNNs in practice in a similar setting to this section.

5.2 Emulating "Weaker" GNNs

In this section we will continue the investigation of GNN expressivity in the bounded degree domain. Instead of looking at injective functions we will evaluate the supposed universality of sum-decomposition by testing if they can emulate weaker aggregation functions such as attention, mean, and max. Theoretically, in the bounded degree domain, sum-decomposition should be able to emulate all these aggregation functions, but can they be learned in practice?

To evaluate this we consider a similar setting as in Section 5.1. The main difference is that we will use randomly initialized GNNs for node label generation instead of relabellings of the WL-test node hashes. A key difference from this experiment and that of the WL-test hashes, is that labels are now correlated, making the problem slightly easier overall. Note that, as in the previous section, this experiment is not intended for testing generalization, we test the ability for the GNNs to generalize in Section 6. In fact Yehudai et al. (2021) show that, under self-distillation alone, GNNs do not generalize to graph size changes, so we should not expect distillations to generalize across architectures.

To gain intuition as to why this is still a useful experiment, assume that we are unaware of the limitations of aggregation functions, e.g. we are unaware that *mean*-aggregation cannot count, or that *max/min*-aggregation treats multisets as sets. Under that assumption, Example 5.1 shows that we will find irreducible error training an **AC-μ** on labels generated by a randomly initialized **AC-Σ** (in expectation).

Example 5.1. Consider the following three configurations:

$a : \langle \texttt{Red}, \{\!| \texttt{Red}, \texttt{Blue} |\!\} \rangle \quad b : \langle \texttt{Red}, \{\!| 2{\cdot}\texttt{Red}, 2{\cdot}\texttt{Blue} |\!\} \rangle \quad c : \langle \texttt{Red}, \{\!| 2{\cdot}\texttt{Red}, \texttt{Blue} |\!\} \rangle$

Convince yourself that a and b and c theoretically are distinguishable by a **AC-Σ** layer (sum aggregation). $\{a, b\}$ and c are distinguishable by **AC-μ** (mean aggregation). None are distinguishable by **SAGE** (max aggregation).

	AC-Σ	GIN	GAT	AC-μ	GCN	SAGE
AC-Σ	85.4±0.9	80.5±1.5	99.4±0.2	92.5±0.7	97.1±0.2	99.7±0.0
GIN	76.2±3.2	82.3±1.4	97.5±3.3	86.3±6.8	96.2±0.6	89.8±7.2
GAT	65.5±2.9	63.1±3.2	94.4±2.6	71.3±3.9	94.0±0.5	92.5±2.5
AC-μ	78.3±1.1	72.7±1.7	98.0±0.7	92.7±0.6	94.8±0.3	99.7±0.1
GCN	64.4±3.4	64.5±3.3	92.3±4.4	63.7±2.4	95.9±0.8	72.9±5.3
SAGE	68.0±2.2	64.9±3.2	95.6±0.6	83.1±1.4	92.7±0.9	98.2±0.4

Table 2: Mean and standard deviation for accuracy when distilling a randomly initialized GNN of the column GNN type into the row GNN type over 5 instantiations. The diagonals are self-distilation. Grayed out cells are impossible to learn in the general case. For improved readability; accuracies from 90-100 are colored green, 80-90 yellow, otherwise red.

Because **AC-**Σ can distinguish a and b, there is a non-zero probability that a randomly initialized **AC-**Σ assigns a and b different labels. But with those labels **AC-**μ must assign the same label to both, leading to irreducible error in expectation. Similarly, **AC-**μ has a non-zero probability of assigning $\{a, b\}$ and c different labels, leading to similar error for **SAGE**.
□

Since we already have theoretical results for the direction described in the example we are interested in the opposite direction: Are there configurations that **GAT** / **AC-**μ / **GCN** / **SAGE** can assign different labels that **AC-**Σ (or **GIN**) cannot *learn* to distinguish in practice.

To test for such configurations (or k-hop neighborhoods) we generate large random graphs with 5 distinct uniformly distributed initial node features, to have a variety of degrees and features. We randomly initialize 5 of each GNN for binary classification and assign their, essentially random, prediction on the random graphs as the node labels. Refer to Appendix B for the full experimental setup. We train on all combinations of GNNs reporting the mean and standard deviation over the 5 initializations in Table 2. Gray cells are not possible in general due to theoretical limitations with counting and multisets.

5.2.1 Discussion

From Table 2 we see that **AC-**Σ generally is able to emulate less powerful GNNs on the training set, showing that there are probably few if any configurations (or 2-hop neighborhoods) that are indistinguishable to it in this setting. **GIN** on the other hand do not perfectly emulate **AC-**μ and **SAGE**, this is interesting because the common factor between **AC-**μ and **SAGE** is that they apply their MLP (resp. weight matrix) to the the previous representation concatenated with the neighborhood aggregation, which is exactly what **GIN** lacks in comparison to **AC-**Σ. This suggests that **GIN** lacks expressivity in practice when node labels rely on a more complex interaction between the previous representation and the neighborhood aggregation. We provide further evidence of

this phenomenon in the experiments on generalization in Section 6. We will also show that, even though **AC-**Σ is able emulate other GNNs, it does not always generalize as well.

6 Logical Expressiveness in Practice

In this section we will attempt to evaluate how well our set of GNNs generalize in practice. In that effort we design three simple logical node classifiers with exact solutions in GNNs. The three classifiers are designed to test different capabilities of GNNs.

We base our initial classifier off of the FOC_2 classifier in Barceló et al. (2020). They show that their node classifier, $\alpha_i(x)$, is not expressible by MPNN, and classify the expressible fragment of FOC_2 as *graded modal logic* (De Rijke, 2000). In graded modal logic all subformulas must be guarded by the edge relation. A formula ϕ can either be a color e.g. $\text{Blue}(x)$, $\text{Red}(x)$, etc. or one of the following:

$$\neg\varphi(x), \quad \varphi(x) \wedge \psi(x), \quad \exists^{\geq N} y(E(x,y) \wedge \varphi(y)) \tag{14}$$

In an effort to test how well GNNs can learn general solutions to counting we define the first logical node classifier β to simply count the number of neighbors with a property:

$$\beta_0(x) := \text{Blue}(x), \quad \beta_{i+1}(x) := \exists^{[N,M]} y(\beta_i(y) \wedge E(x,y)) \tag{15}$$

Where we define $\exists^{[N,M]} y(\psi) = \exists^{\geq N} y(\psi) \wedge \neg\exists^{\geq M+1} y(\psi)$. Notice how the formula β is recursively defined such that we can make it arbitrarily hard. $\beta_1(v)$ is true if v has between N and M blue neighbors. $\beta_2(v)$ is true if v has between N and M neighbors who have between N and M blue neighbors and so on. As mentioned in Section 4.4, we only expect GNNs with sum aggregations to perform well. We show in Lund (2023) that low-dimensional exact solutions exist for **GIN** and **AC-**Σ of all β_i.

We design the next logical classifier in an effort to test other GNNs along with the ability for sum-decomposition to learn general solutions for distributional aggregations. To describe distributional aggregation we must modify the grammar to have *ratio of degree* existential counting qualifiers. We now only allow counting qualifiers of the form $\exists^{\geq \lceil N/M D(x) \rceil} y(\psi)$ where $D(x)$ is the degree of the node. This lets us define a distributional node classifier γ:

$$\gamma_0(x) := \text{Blue}(x), \quad \gamma_{i+1}(x) := \exists^{\geq \lceil N/M D(x) \rceil} y(\gamma_i(y) \wedge E(x,y)) \tag{16}$$

If we temporarily fix $N/M = 1/2$ we have that $\gamma_1(v)$ is true if half or more of v's neighbors are blue. $\gamma_2(v)$ is true if half or more of v's neighbors have half or more blue neighbors, etc. While we break the syntax of graded modal logic, we show in Lund (2023) that exact solutions still exist for **GIN** and **AC-**Σ, and the same method can be used for exact solutions for **AC-**μ.

Finally we design a classifier that can be solved with max aggregation too. The classifier is very simple as we cannot use counting qualifiers. We define δ as seen below:

$$\delta_0(x) := \text{Blue}(x), \quad \delta_{i+1}(x) := \neg\delta_i(x) \wedge \exists y(\delta_i(y) \wedge E(x, y)) \tag{17}$$

We have that $\delta_1(v)$ is true if v is not blue and has a blue neighbor. $\delta_2(v)$ is true if v has a non-blue neighbor with a blue neighbor and is blue or does not have a blue neighbor. Again exact solutions exist[3].

With these simple formulas defined we can finally evaluate how well GNNs generalize on them. We generate a training set of random graphs from size 50 to 100 nodes with 5 distinct uniformly distributed initial node features (One being Blue). Node labels are assigned by running implementations of the node classifiers on the random graphs. Because the graphs are Erdős–Rényi graphs (Erdős et al., 1960) with a fixed edge probability, the edge distribution changes as the graph size increases. We test the classifiers on test sets consisting of graphs up to 300 nodes in size and plot the mean accuracy over 5 runs. Results for counting (β) can be seen in Figure 6. Distributions (γ) in Figure 7. Non-counting (δ) in Figure 8. The training graph size is shown as a grey band on all plots. Refer to Appendix C for the full experimental setup.

6.1 Discussion

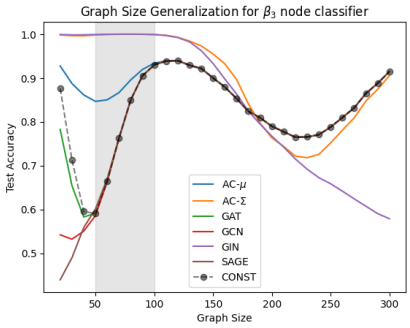
For the counting classifier β in Figure 6 we see that both **GIN** and **AC-**Σ are able to learn the counting classifiers to a perfect accuracy within the training set. Unfortunately, the performance does not generalize to large graph sizes for either of them, even though the problem is pretty simple. Both even regress below the line for predicting a constant value.

For the distributional node classifier γ in Figure 7, both sum-decomposition GNNs perform surprisingly well, even outperforming **AC-**μ with mean aggregation most of the time. On the harder problem γ_5 they do both eventually degrade in performance, whereas simple methods like **GCN** and **SAGE**, without exact solutions, maintain their, albeit lower, performance.

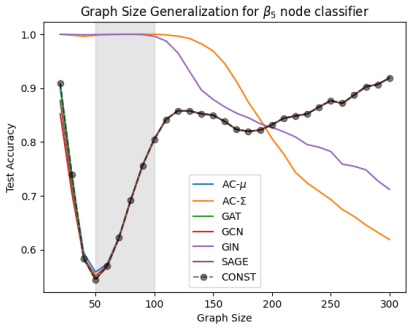
Finally for the non-counting node classifier δ in Figure 8, we see that most GNNs achieve near-perfect accuracy in the training distribution. Of those GNNs only **GIN** and **AC-**Σ drop in performance as the graph size changes suggesting that **SAGE** and **AC-**μ learn more general solutions in this setting. **GIN** drops off much more rapidly, giving for credence to the hypothesis that it has trouble disentangling its own representation from that of its neighbors when there are complex interactions between them.

Generally, **GAT** performs poorly in these experiments. It may be the case that logical classifiers do not lend themselves to an attentional aggregation scheme.

[3]For β_{i+1} one can simply normalize the β_i contribution from the aggregation and negate the β_i contribution from the previous representation and do a logical and between the two bits which is the truth value of β_{i+1}



(a) Graph size generalization results for the counting node classifier β_3 from Equation 15



(b) Graph size generalization results for the counting node classifier β_5 from Equation 15

Figure 6: Mean test set accuracy over 5 runs. In both plots the grey band indicates the graph sizes during training. The dotted line ("CONST") shows the accuracy of predicting a constant value.

Overall, none of the GNNs tested consistently generalize well, this suggests that the choice of GNN should be problem specific, as the most expressive GNNs are not always the best choice.

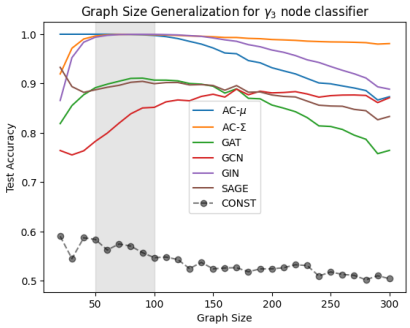
In the following section we investigate this further by evaluating all GNNs on real world datasets with improved expressivity from the **ESAN** framework.

7 Higher Order GNNs With Weaker Encoders

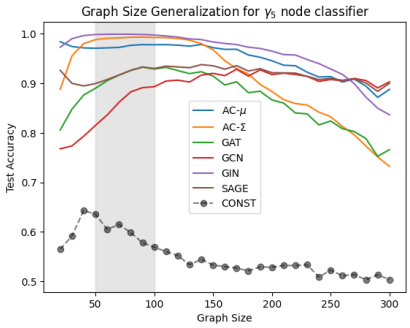
Based on the results in the previous section we find that the selection of GNN is problem specific. To evaluate this outside of a synthetic setting, we will in this section consider two real world datasets: MUTAG and PTC. We aim to see whether the choice of GNN effects the performance of the **ESAN** framework on real datasets. The usage of **ESAN** is mainly motivated by the observation that subgraph selection policies with node marking allow distributional aggregators to count, showing that it can improve the expressivity of GNNs without sum-decomposition. This can be seen as an effort to motivate future work using GNN encoders to consider non sum-decomposition GNNs when evaluating their methods.

MUTAG Debnath et al. (1991) is a dataset of molecyles. Nodes are atoms and edges are chemical bonds. Graph are labelled by their mutagenic effect on a bacterium. The dataset consists of 188 graphs with 7 discrete node labels. There are 27163 nodes and 148100 edges in total in the dataset.

PTC (Helma et al., 2001) is a dataset of chemical compounds. The compounds are labelled with the carcinogenicity for rats. The dataset consists of

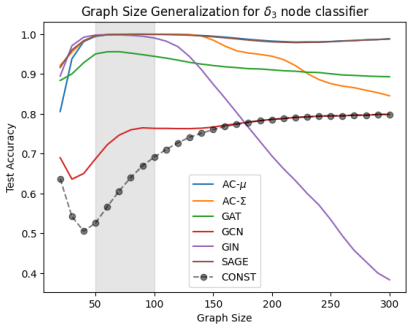


(a) Graph size generalization results for the distributional node classifier γ_3 from Equation 16

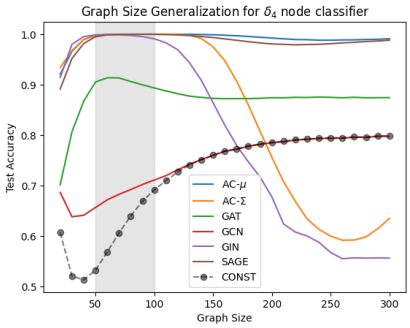


(b) Graph size generalization results for the distributional node classifier γ_5 from Equation 16

Figure 7: Mean test set accuracy over 5 runs. In both plots the grey band indicates the graph sizes during training. The dotted line ("CONST") shows the accuracy of predicting a constant value.



(a) Graph size generalization results for the non-counting node classifier δ_3 from Equation 17



(b) Graph size generalization results for the non-counting node classifier δ_4 from Equation 17

Figure 8: Mean test set accuracy over 5 runs. In both plots the grey band indicates the graph sizes during training. The dotted line ("CONST") shows the accuracy of predicting a constant value.

344 graphs with 19 discrete node labels. There are 8792 nodes and 17862 edges in total in the dataset.

We evaluate all GNNs from Section 3.2: **GIN**, **AC-**Σ, **AC-**μ, **GAT**, **GCN**, **GraphSAGE**. Along with three subgraph selection policies from Section 3.3: Edge-deleted (ED), Node-deleted (ND), Ego-networks (EGO+).

We did not have enough VRAM to run **GAT** with the ED policy, so we cannot report results for that configuration.

We use the same training procedure as in Bevilacqua et al. (2022) and Xu et al. (2019), reporting results for 10-fold cross validation in Table 3. Due to compute limitations we train for less epochs than Bevilacqua et al. (2022) and do less hyperparameter search. Refer to Appendix D for the full experimental setup.

Method ↓ / Dataset →	MUTAG	PTC
SoTA	92.7±6.1	68.2±7.2
GIN (Xu et al., 2019)	89.4±5.6	64.6±7.0
DS-GNN (GIN) (Bevilacqua et al., 2022)	91.0±4.8	68.7±7.0
GIN	89.4±5.3	64.9±4.9
DS-GNN (GIN) (ED)	89.9±7.4	66.2±7.8
DS-GNN (GIN) (ND)	89.8±6.2	65.6±6.8
DS-GNN (GIN) (EGO+)	89.9±5.1	64.2±3.2
AC-Σ	88.9±5.0	62.8±9.6
DS-GNN (AC-Σ**) (ED)**	87.8±3.3	63.9±5.6
DS-GNN (AC-Σ**) (ND)**	88.8±4.5	64.8±7.6
DS-GNN (AC-Σ**) (EGO+)**	89.9±5.0	66.5±8.4
AC-μ	84.1±3.9	64.5±4.3
DS-GNN (AC-μ**) (ED)**	86.7±7.1	66.2±4.7
DS-GNN (AC-μ**) (ND)**	85.6±4.9	65.4±6.0
DS-GNN (AC-μ**) (EGO+)**	**91.5±4.9**	64.2±5.8
GAT	83.1±8.7	65.4±4.0
DS-GNN (GAT) (ED)	-	-
DS-GNN (GAT) (ND)	84.6±6.8	65.4±6.4
DS-GNN (GAT) (EGO+)	89.4±4.8	66.0±9.1
GCN	90.4±3.1	63.9±7.0
DS-GNN (GCN) (ED)	90.9±5.4	64.6±6.5
DS-GNN (GCN) (ND)	88.3±3.1	62.2±7.0
DS-GNN (GCN) (EGO+)	90.5±4.6	66.5±6.4
GraphSAGE	86.7±8.2	65.7±5.2
DS-GNN (GraphSAGE) (ED)	85.6±5.9	65.7±5.7
DS-GNN (GraphSAGE) (ND)	85.1±5.8	63.7±6.0
DS-GNN (GraphSAGE) (EGO+)	90.0±6.4	65.7±8.1

Table 3: Results on MUTAG and PTC datasets for **ESAN** with different graph encoders. Mean and standard deviation are computed from a 10-fold validation

7.1 Discussion

Looking at Table 3 all GNNs are able to perform better than the base **GIN** from Xu et al. (2019) in both datasets with some subgraph selection policy, but most are unable to do so without it. This suggests that **ESAN** is able to boost the expressivity of all used MPNNs. Furthermore we find that, even with limited finetuning compared to Bevilacqua et al. (2022), **DS-GNN (AC-μ) (EGO+)** is able to outperform the best result from Bevilacqua et al. (2022) on MUTAG. This is further evidence that one should not always select the most expressive GNN for all problems. The fact that many GNNs are applicable also highlight that the hunt for good MPNNs is not necessarily over, as there is still room for less expressive, better generalizing models. Our results here go to show that it will be beneficial for future subgraph or graph structure frameworks to evaluate on weaker GNNs such as **AC-μ** as well.

8 Conclusion

Throughout our experiments we have shown that the theoretical results on sum-decomposition put hard limits on the practicality of using universal MPNNs. We have shown that universal MPNNs cannot learn injective functions such as the WL-test in practice. We find that universal MPNNs are able to emulate most less expressive MPNNs, but also some evidence that **GIN** struggles to disentangle its own representation from that of its neighborhood when there are complex interactions between them. On synthetic experiments on logic node classifiers we find that all MPNNs are unable to generalize to larger graphs in all settings. We find that some MPNNs are better suited for different tasks, emphasising the need to explore different MPNNs. Finally, we show that MPNNs also benefit from structural information in GNNs such as **ESAN**. We find that all MPNNs improve in performance on real world datasets when applied to the **ESAN** framework. We even find that an MPNN with mean aggregation can outcompete universal MPNNs on real world datasets given the structural information. Overall we propose that the current practice of only evaluating methods that improve MPNNs on universal MPNNs should be modified to consider a small set of less powerful MPNNs as well.

Future Work We only test the least expressive version of **ESAN** in our experiments. Applying the more expressive **DSS-GNN** could lead to more insights into how less powerful MPNNs can be used. We know that the theoretical expressivity of **ESAN** with a WL-1 encoder is strictly greater than WL-1, but we have no theoretical results for encoders that are less powerful, such as mean aggregating MPNNs. We highlight situations where they can count, but we need a theoretical analysis to determine where they lie in the WL-hierarchy.

References

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J., and Silva, J. P. (2020). The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. (2022). Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*.

Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. (2021a). Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640.

Bodnar, C., Frasca, F., Wang, Y., Otter, N., Montufar, G. F., Lio, P., and Bronstein, M. (2021b). Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pages 1026–1037. PMLR.

Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. (2022). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668.

Cai, J.-Y., Fürer, M., and Immerman, N. (1992). An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410.

Chen, Z., Chen, L., Villar, S., and Bruna, J. (2020). Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

De Rijke, M. (2000). A note on graded modal logic. *Studia Logica*, 64(2):271–283.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29.

Erdős, P., Rényi, A., et al. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Grohe, M. and Otto, M. (2015). Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Helma, C., King, R. D., Kramer, S., and Srinivasan, A. (2001). The Predictive Toxicology Challenge 2000–2001 . *Bioinformatics*, 17(1):107–108.

Lund, S. (2023). Pre-specialization: Evaluating graph neural networks under edge distribution shifts.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019). Provably powerful graph networks. *Advances in neural information processing systems*, 32.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609.

Pellegrini, G., Tibo, A., Frasconi, P., Passerini, A., and Jaeger, M. (2021). Learning aggregation functions. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2892–2898. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Sato, R., Yamada, M., and Kashima, H. (2021). Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM.

Schöning, U. (1988). Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*. accepted as poster.

Wagstaff, E., Fuchs, F., Engelcke, M., Posner, I., and Osborne, M. A. (2019). On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494. PMLR.

Weisfeiler, B. and Leman, A. (1968). The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16.

Welling, M. and Kipf, T. N. (2016). Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

Yehudai, G., Fetaya, E., Meirom, E., Chechik, G., and Maron, H. (2021). From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pages 11975–11986. PMLR.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. *Advances in neural information processing systems*, 30.

Zhang, B., Luo, S., Wang, L., and He, D. (2023). Rethinking the expressive power of GNNs via graph biconnectivity. In *International Conference on Learning Representations*.

	WL Step 1	WL Step 2	WL Step 3
1 Color	14	4.000	80.000
2 Colors	116	90.000	350.000

Table 4: Unique hashes per WL step in the training sets.

A Experimental Setup: Emulating The WL Test

We generate 5000 random undirected Erdős–Rényi Erdős et al. (1960) graphs of size 20-30 nodes (uniform) as the training set. Every edge (undirected) is included in the graph with a probability of $p = 0.1$. Table 4 shows the number of unique hashes in the training set (before any relabeling). For ***1 Color*** all nodes are initialized with a feature vector of $\mathbf{x} = [1]$. For ***2 Colors*** one of two colors is chosen uniformly for each node and is one-hot encoded as the feature vector of the node.

For each step of the WL test the 5 relabelling functions for node labels are generated from taking the ith bit of the WL steps hash. We use NetworkX (Hagberg et al., 2008) to generate the subgraph hashes for each node.

We train all GNNs with layers equal to the WL step. E.g. WL step 2 uses a 2 layer **GIN** and $AC - \Sigma$. In all GNNs we have 4-layer MLPs. We train with a batch size of 256 for up to 500 epochs with early stopping in case of no improvement for 10 epochs. Learning rate is initially 0.01 but we halve it every 10th epoch (Step Decay). In Table 1 we train with a hidden dimension of $h = 64$. For results in Figure 5 we train with hidden dimension $h \in [8, 16, 32, 64, 128, 256, 512, 1024]$.

B Experimental Setup: Emulating "Weaker" GNNs

We generate 5000 random undirected Erdős–Rényi Erdős et al. (1960) graphs of size 50-100 nodes (uniform) as the training set. Every edge (undirected) is included in the graph with a probability of $p = 0.1$. All nodes are initialized with one of 5 colors, chosen uniformly for each node and is one-hot encoded as the feature vector of the node.

To generate labels we randomly initialize GNNs with $l = 2$ GNN layers, 4-layer MLPs (if they have MLPs), and a hidden dimension of $h = 64$. We randomly initialize 5 of each GNN with those hyperparameters and evaluate them on the test set. We prune degenerate GNNs (ones predicting the same value always, or almost always) by checking if the label frequency is below 20% or above 80%, if so we reinitialize and retry. We always pick the label with the highest logits instead of randomly sampling.

For training all GNNs with MLPs have 4-layer MLPs. We train with a batch size of 256 for up to 500 epochs with early stopping in case of no improvement for 10 epochs. Learning rate is initially 0.01 but we halve it every 10th epoch (Step Decay). We hyperparameter search for hidden dimension $h \in [64, 128]$

	$i = 1$	$i \in [2, \dots 5]$
β	$[1, 3]$	$[2, 3]$
γ	$1/5$	$1/2$

Table 5: Constants used in counting quantifiers for β_i and γ_i

and number of GNN layers $l \in [2, 3]$, picking the best performing to aggregate over for Table 2.

We use Weights & Biases (Biewald, 2020) to track experiments and perform parameter sweeps.

C Experimental Setup: Logical Node Classifiers

We generate 5000 random undirected Erdős–Rényi Erdős et al. (1960) graphs of size 50-100 nodes (uniform) as the training set. Every edge (undirected) is included in the graph with a probability of $p = 0.1$. All nodes are initialized with one of 5 colors, chosen uniformly for each node and is one-hot encoded as the feature vector of the node. The first color is considered Blue. The constants used for β_i and γ_i are shown in Table 5

For test sets we generate 1000 random graphs for every size from 10-300 in 10 step intervals ([10, 20, 30, ...]). We label the training and tests sets with the node classifiers. For all ϕ_i experiments we use i-layer GNNs with a hidden dimension of $h = 64$. We train with a batch size of 256 for up to 500 epochs with early stopping in case of no improvement for 10 epochs. Learning rate is initially 0.01 but we halve it every 10th epoch (Step Decay). We average the test set accuracy over 5 runs for each GNN.

We use Weights & Biases (Biewald, 2020) to track experiments and perform parameter sweeps.

Figure 9 shows the cumulative probability for each degree for a node based as the size of graphs increases. The gray band is the training graph size distribution.

D Experimental Setup: Higher Order GNNs With Weaker Encoders

We use the publicly available Bevilacqua et al. (2022) codebase for running the experiments. We implement **AC-Σ**, **AC-μ** and use the PyTorch Geometric (Fey and Lenssen, 2019) implementation of **GAT** and **GraphSAGE** in the experiments. We use Weights & Biases (Biewald, 2020) to track experiments and perform parameter sweeps.

Due to compute limitations we train for 200 epochs where **ESAN** (Bevilacqua et al., 2022) train for 350 epochs. We use a fixed embedding dimension of 32 whereas **ESAN** sweep over $h \in [16, 32]$. We use a fixed batch size of 64 whereas

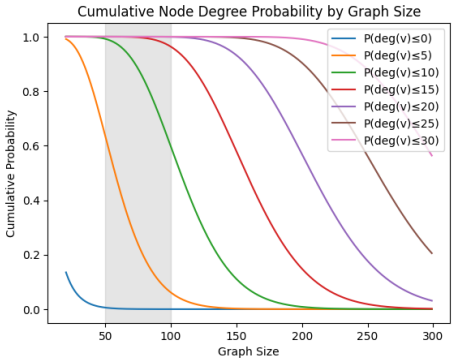


Figure 9: Distribution of node degrees as the graph size increases. As the graph size increases, it is less likely for a node to have few neighbors.

ESAN sweep over $b \in [32, 128]$. We sweep over learning rates $lr \in [0.01, 0.001]$, as does **ESAN**. We use a fixed DeepSets embeddings size of 64 whereas **ESAN** sweep over $h_D \in [32, 64]$