

# **A Theoretical and Empirical Investigation into the Equivalence of Graph Neural Networks and the Weisfeiler-Leman Algorithm**

From the faculty of Mathematics, Physics, and Computer Science approved for the purpose of obtaining  
the academic degree of Bachelor of Sciences.

**Eric Tillmann Bill**

Supervision:  
Prof. Dr. rer. nat. Christopher Morris  
Informatik 6  
RWTH Aachen University

July 8, 2023



# **Abstract**

  Lorem Ipsum :D

## **Acknowledgements**

Thanks to everyone :D

# Contents

1.	Introduction . . . . .	1
1.1.	Motivation . . . . .	1
1.2.	Research Questions & Contributions . . . . .	2
1.3.	Methology . . . . .	2
1.4.	Outline . . . . .	3
2.	Background and Related Work . . . . .	3
2.1.	Weisfeiler-Leman Algorithm . . . . .	3
2.2.	Graph Neural Networks . . . . .	4
<b>I.</b>	<b>Theoretical Equivalence</b>	<b>7</b>
3.	Preliminaries . . . . .	8
3.1.	General Notation . . . . .	8
3.2.	Graphs . . . . .	8
3.3.	Weisfeiler and Leman Algorithm . . . . .	9
3.4.	1-WL+NN . . . . .	11
3.5.	Graph Neural Networks (Message Passing) . . . . .	11
4.	Theoretical Connection . . . . .	13
4.1.	Proof of Theorem 11 . . . . .	14
4.2.	Proof of Theorem 12 . . . . .	17
<b>II.</b>	<b>Empirical Testing</b>	<b>21</b>
5.	Testing Configuration . . . . .	22
5.1.	Datasets . . . . .	22
5.2.	Choice of Models . . . . .	25
5.3.	Experimental Setup . . . . .	26
6.	Empirical Testing . . . . .	28
6.1.	Fixed 1-WL Iterations: Tradeoff between Expressiveness and Generality	30
6.2.	Difference in Learning behavior . . . . .	31
6.3.	Aggregation Differnces . . . . .	33
6.4.	GNN Approximating 1-WL . . . . .	33
6.5.	Preprocessing Data for GNNs . . . . .	33
7.	Discussssion . . . . .	33
7.1.	Learned Lessons . . . . .	33
7.2.	Future Work . . . . .	33
8.	Conclusion . . . . .	33
	<b>Bibliography</b>	<b>38</b>

<b>A. Appendix Part I</b>	<b>42</b>
1. Figures and graphs . . . . .	42
2. Proofs . . . . .	42
2.1. Lemma 20: Composition Lemma for 1-WL+NN . . . . .	42
<b>B. Appendix Part II</b>	<b>44</b>
1. Definitions . . . . .	44
1.1. Definition of the normalized Shannon Index . . . . .	44
1.2. Theoretical Maximum Accuracy Analysis . . . . .	44
1.3. Hyperparameter Configuration and Optimization . . . . .	45
2. Visualization of the Approximation Performance of GNNs . . . . .	46

# 1. Introduction

This work aims to shed light on the representations learned by Graph Neural Networks (GNNs). In this section, we will discuss the prevalence of graphs and the crucial role GNNs play in analyzing them. We will delve into the methods we use to gain insights and highlight the significance of our approach. Lastly, we will provide an overview of the structure of this work.

## 1.1. Motivation

Graphs are ubiquitous in various fields of life. Despite not always being explicitly identified as such, the graph data model’s flexibility and simplicity make it an effective tool for modeling a diverse range of data. Examples of graph modeling applications include unexpected instances, such as modeling text or images as a graph, as well as more complex instances like chemical molecules, citation networks, or connectivity encodings of the World Wide Web Morris et al. [2020], Scarselli et al. [2009].

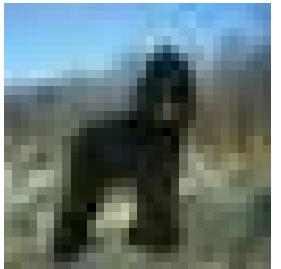
Although machine learning has achieved remarkable success with image classification (e.g., Zoph et al. [2018], He et al. [2016]) and text generation (e.g., Radford et al. [2019], Brown et al. [2020]) in the last decade, the lack of a significant breakthrough in machine learning for graphs can be attributed to the graph data model’s inherent flexibility and simplicity. While, for example, an image classifier constrains its input data to be a grid-like image or a text generator expects its input to be a linear sequence of words, machine learning models working on graphs cannot leverage any constraints on the format or size of their input graphs without limiting their generality.

To put this flexibility of the graph data model into perspective and give an idea of how ubiquitous graphs are in various fields, we refer back to the examples of image classifiers and text generators and demonstrate how seemingly natural the graph data model can encode their input data. For example, images can be encoded by a graph, such that each pixel of the image corresponds to a node in the graph holding its color value, and each node is connected to its neighboring pixel nodes. Similarly, for sequential data like text files, one can encode a directed graph where each word in this file is represented as a node with the word as a feature and connected outgoingly to the next following word. See Figure 1 for an illustrative example of these encodings.

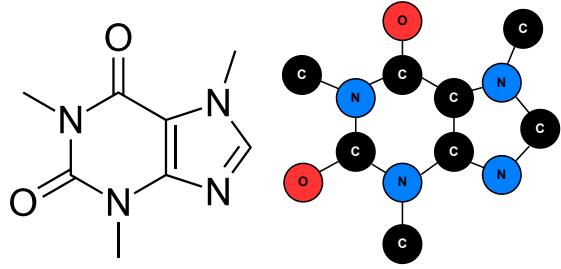
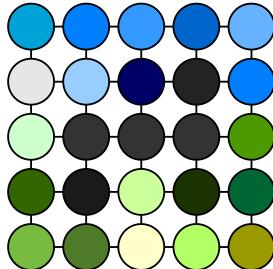
In recent years, a significant amount of research has been conducted to investigate Graph Neural Networks (GNNs). Among the most promising approaches are those utilizing the message-passing architecture, which was introduced by Scarselli et al. [2009] and Gilmer et al. [2017]. Empirically, this framework has demonstrated strong performance across various applications Kipf and Welling [2017], Hamilton et al. [2017], Xu et al. [2019]. However, its expressiveness is limited, as has been proved by the works of Morris et al. [2019], as well as Xu et al. [2019]. These works establish a connection to the commonly used Weisfeiler-Leman<sup>1</sup> algorithm (1-WL), originally proposed by Weisfeiler and Leman [1968] as a simple heuristic for the graph isomorphism problem. In particular, it has been proven that a GNN based on the message-passing architecture can, at most, be as good as the 1-WL algorithm in distinguishing non-isomorphic graphs. Furthermore, the 1-WL method demonstrates numerous similarities with the fundamental workings of the GNN architecture. It is, therefore, commonly believed

---

<sup>1</sup>Based on <https://www.iti.zcu.cz/wl2018/pdf/leman.pdf>, we will use the spelling “Leman” here, as requested by A. Leman, the co-inventor of the algorithm.

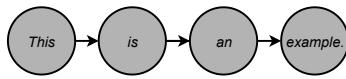


(a) Example for an image of a dog.<sup>2</sup>



(b) Example for the molecule caffeine.<sup>3</sup>

<b>Example.txt</b>
<i>This is an example.</i>



(c) Example for a text file.

Figure 1.: Here are some examples of how graphs can be used to encode information in a variety of domains. Please note that these examples are just a sample, and in actual practice, more detailed encodings are usually utilized to capture additional information.<sup>4</sup>

that both methods are, to some extent, equivalent in their capacity to capture information in a graph.

## 1.2. Research Questions & Contributions

Test

## 1.3. Methodology

In this work, we introduce a novel framework, which we coined “1-WL+NN,” which involves applying the 1-WL algorithm to an input graph and further processing the resulting information using a feedforward neural network. Thereby, we obtain a trainable framework suited for all kinds of graph-related tasks, such as graph classification, node regression, and more. We will prove later on that both frameworks, 1-WL+NN and GNN, are theoretically equivalent, such that each function computed by a 1-WL+NN can also be computed by a GNN model and vice versa. With this framework in hand, we can investigate the representations learned by a GNN.

The interesting property of this framework compared to GNNs, which is also the original idea that inspired this work, is the fundamental difference in how both frameworks learn and optimize themselves when applied to a specific task. Take, for example, an arbitrary classification task. While the first part of a 1-WL+NN model starts by applying the 1-WL algorithm to its input graph and retrieves a highly informative representation of this graph, the second part, the learnable feedforward neural network, must find common patterns in this very detailed representation that coincides with the classes of the task, such that the model

<sup>2</sup>The image is from the CIFAR-10 collection made available by Krizhevsky et al. [2009].

<sup>3</sup>The illustration of the skeletal formula of caffeine is taken from <https://commons.wikimedia.org>.

<sup>4</sup>All graphics were created using the free open source platform <https://www.draw.io>.

Footnotes  
are  
wrong!

more  
colorful  
and  
downscaled  
example  
:(

makes good predictions. In comparison, while a GNN is theoretically as expressive as the 1-WL algorithm, we imagine that such a model first leans to find common patterns as early as possible in the input graph and will drop irrelevant information as soon as possible before making a class prediction.

To put both learning behaviors into perspective: while the 1-WL+NN is given a maximally informative representation and needs to find the important information to make a good prediction, a GNN works the other way around and needs to learn how it constructs its own informative representation of the input graph and how to leverage this information for making a prediction. Therefore, we expect 1-WL+NN models to perform sufficiently well on their training data but expect poor generalization capabilities due to the too-informative representations computed by the 1-WL algorithm. In contrast, we expect more promising generalization capabilities of GNNs in comparison since they optimize for the best representation of their input graphs and might leverage this information more efficiently.

We will use this novel framework and various empirical experiments to compare both frameworks on multiple datasets to establish a deeper understanding of the representations learned by GNNs.

#### 1.4. Outline

For the ease of readability, we split this work into two parts. The first part investigates and establishes the theoretical equivalence between the frameworks of 1-WL+NN and GNNs. In contrast, the second part discusses our different experiments and their empirical results and insights into GNNs.

In detail, this work starts with Section 2, where we discuss related work, milestones in GNNs over the past decade, essential properties of the 1-WL algorithm, and a subset of interesting connections between GNNs and the 1-WL algorithm. Afterward, we will start with Part I, which begins with Section 3. Here, we will introduce formal definitions for both frameworks, as well as a set of notations we will use throughout the theoretical part. Afterward, in Section 4, we will introduce four theorems that each present a connection between both frameworks and combined prove the equivalence of both frameworks. Finally, we will prove each theorem individually after another in corresponding subsections.

The second part will deal with ...

## 2. Background and Related Work

In this section, we will briefly introduce the foundation of our research by explaining the origins of each method, mentioning important recent advances, and providing a brief overview of the connections between them.

### 2.1. Weisfeiler-Leman Algorithm

The (1-dimensional) Weisfeiler-Leman algorithm (1-WL), proposed by Weisfeiler and Leman [1968], was initially designed as a simple heuristic for the *graph isomorphism problem*, but due to its interesting properties, its simplicity, and its good performance, the 1-WL algorithm gained a lot of attention from researchers across many fields. One of the most noticeable properties is that the algorithm color codes the nodes of the input graph in such a way that in each iteration, each color encodes a learned local substructure.

It works by coloring all nodes in each iteration the same color that fulfills two properties: 1. the nodes already share the same color, and 2. the count of each color of their direct neighbors is equal. The algorithm continues as long as the number of colors changes in each iteration. For determining whether two graphs are non-isomorphic, the heuristic is applied to both graphs simultaneously. It concludes that the graphs are non-isomorphic as soon as the number of occurrences of a color differs between them. We present a more formal definition of the algorithm in the following part in subsection 3.3.

Since the *graph isomorphism problem* is difficult to solve due to the best known complete algorithm only running in deterministic quasipolynomial time (Babai [2016]), the 1-WL algorithm, running in deterministic polynomial time, cannot solve the problem completely. Moreover, Cai et al. [1992] constructed counterexamples of non-isomorphic graphs that the heuristic fails to distinguish, e.g., see Figure 3. However, following the work of Babai and Kucera [1979], this simple heuristic is still quite powerful and has a very low probability of failing to distinguish non-isomorphic graphs when both graphs are uniformly chosen at random as the number of nodes tends to infinity.

To overcome the limited expressiveness of the 1-WL algorithm, it has been generalized to the  $k$ -dimensional Weisfeiler-Leman algorithm ( $k$ -WL) by Babai [1979, 2016], as well as Immerman and Lander [1990]<sup>5</sup>. This version works with  $k$ -tuples over the  $k$ -ary Cartesian product of the set of nodes. Interestingly, this created a hierarchy for the expressiveness of determining non-isomorphism, such that for all  $k \in \mathbb{N}$  there exists a pair of non-isomorphic graphs that can be distinguished by the  $(k + 1)$ -WL but not by the  $k$ -WL (Cai et al. [1992]).

## 2.2. Graph Neural Networks

The utilization of machine learning techniques, previously proven effective in various domains, for graph analysis has been a well-established topic in the literature for several decades. However, researchers faced challenges in effectively adapting these techniques to graphs of diverse sizes and complexities in the early stages. Notably, the seminal works by Sperduti (1997), Scarselli (2008), and Micheli (2009) emerged as prominent examples of successful applications in this regard.

The idea of leveraging machine learning techniques, previously proven effective in various domains, for graph-related tasks has been a well-established topic in the literature for the past decades. However, in the early stages, researchers faced challenges in effectively adapting these techniques to work on graphs of arbitrary sizes and complexities. Notably, the works by Sperduti and Starita [1997], Scarselli et al. [2008], and Micheli [2009] were the first prominent examples of successful applications in this regard.

However, it was not until the emergence of more advanced models that the scientific community truly recognized the significance and potential of Graph Neural Networks (GNNs). Noteworthy among these advancements were the work of Duvenaud et al. [2015], who introduced a differentiable approach for generating unique fingerprints of arbitrary graphs, as well as Li et al. [2015], who applied gated recurrent units to capture graphs of various sizes, while Atwood and Towsley [2016] utilized diffusional convolutions for the same purpose. Of particular significance, however, were the contributions of Bruna et al. [2013], Defferrard et al. [2016] and Kipf and

---

<sup>5</sup>In Babai [2016] on page 27, László Babai explains that he, together with Rudolf Mathon, first introduced this algorithm in 1979. He adds that the work of Immerman and Lander [1990] introduced this algorithm independently of him.

Welling [2017], which extended the concept of convolution from its traditional application on images to the domain of arbitrary graphs.

After the early success of these GNNs, Gilmer et al. [2017] introduced a unified architecture for GNNs. The authors observed a recurring pattern in how information is exchanged and processed among many of these works, including many mentioned in the paragraph above. Leveraging these observations, Gilmer et al. [2017] devised the message-passing architecture as a generalized framework for GNNs. Models using this architecture can be referred to as Message-Passing-Neural-Network (MPNN); however, throughout this thesis, we will use the term GNN and MPNN interchangeably, as the focus of this thesis is solely on the message-passing architecture. This architecture uses the input graph as its basis for computation and computes new node features for the graph in each layer. Each new node feature is derived by aggregating the nodes and neighboring node features. After applying each layer of a GNN model, a representation of the entire graph is obtained by applying a pooling function (see YingMorris2018). This representation is then further processed by common machine learning techniques like a multilayer perceptron for the final output. We will present a more formal definition of this architecture in the following part; however, important to note is that the information exchange in the graph across nodes is limited to a one-hop neighbor per layer.

With this general framework and the empirical success of some models using this message-passing architecture, the question of how expressive models based on this architecture can be gained a lot of attention in the scientific community. Many papers immediately established connections to the 1-WL algorithm, among the most prominent being Morris et al. [2019] and Xu et al. [2019]. These connections seem natural, as both methods share similar properties in terms of how they process graph data. Most strikingly, both methods never change the graph structurally since they only compute new node features in each iteration. Moreover, both methods use a one-hop neighborhood aggregation as the basis for computing the new node feature. Following this intuition, Morris et al. [2019] as well as Xu et al. [2019] showed that the expressiveness of GNN is upper-bounded by the 1-WL in terms of distinguishing non-isomorphic graphs. Moreover, Morris et al. [2019] proposed a new  $k$ -GNN architecture that operates over the set of subgraphs of size  $k$ . Interestingly, Geerts [2020] as well as Grohe [2017] have shown that the proposed hierarchy over  $k \in \mathbb{N}$  is equivalent to the  $k$ -WL hierarchy in terms of its ability to distinguish non-isomorphic graphs, i.e., if there is a  $k$ -GNN that can distinguish two non-isomorphic graphs, it is equivalent to say that the  $k$ -WL algorithm can also distinguish these graphs.

Although there are other modifications of the message-passing architecture besides the theoretical concept of  $k$ -GNN to increase the expressiveness of GNNs in terms of distinguishing non-isomorphism, e.g., using node identifiers Vignac et al. [2020], adding random node features Sato et al. [2021], Abboud et al. [2020], adding directed flows Beaini et al. [2021] and many more. Relatively few works have been published that attempt to understand the representation learned from a standard GNN.

Notable works include Nikolentzos et al. [2023b], where the author, in addition to the normal learning process, optimized GNNs to preserve a notion of distance in their representation and examined the effectiveness of GNNs in utilizing such representations. However, their insights can only be applied to these specially trained GNNs, not standard ones. In another publication, Nikolentzos et al. [2023a] presented mathematical proof and empirical confirmation showing how much structural information is encoded by modern GNN models. The research highlights that GNN models like DGCNN (Zhang et al. [2018]) and GAT (Veličković et al. [2017]) encode all nodes with the same feature vector, while in contrast, models like GCN (Kipf and Welling

[2017]) and GIN (Xu et al. [2019]) encode nodes after  $k$  layer of message-passing with features that relate with the number of walks of length  $k$  form each node, disregarding the local structure within the nodes are contained.

**Part I.**

**Theoretical Equivalence**

### 3. Preliminaries

First, we introduce a couple of notions and definitions that will be used throughout this thesis. In particular, the definitions will be crucial in the following section containing the proofs. We start with general notations, introduce a general graph definition, and familiarize the reader with the Weisfeiler-Leman algorithm. We will introduce each framework independently, first the 1-WL+NN and then GNN. In the end, we will briefly introduce important properties of collections of functions computed by both methods.

#### 3.1. General Notation

Let  $\mathbb{N}$  denote the set of natural numbers such that  $\mathbb{N} := \{0, 1, 2, \dots\}$ . By  $[n]$ , we denote the set  $\{0, \dots, n\} \subset \mathbb{N}$  for each  $n \in \mathbb{N}$ . Further, with  $\{\dots\}$  we denote a multiset formally defined as a 2-tuple  $(X, m)$ , where  $X$  is a set of all unique elements and  $m : X \rightarrow \mathbb{N}_{\geq 1}$  a mapping that maps each element in  $X$  to the number of its occurrences in the multiset.

#### 3.2. Graphs

We will briefly introduce a formal definition for graphs and coloring on graphs.

**Definition 1** (Graph). A graph  $G$  is defined as a 3-tuple denoted by  $G := (V, E, l)$ . This tuple consists of a set of nodes  $V \subset \mathbb{N}$ , a set of edges  $E \subseteq V \times V$ , and a labeling function  $l : M \rightarrow \Sigma$ . The domain  $M$  of the labeling function can be either  $V$ ,  $V \cup E$ , or  $E$ , and the codomain  $\Sigma$  is a finite alphabet with  $\Sigma \subset \mathbb{N}^k$ , where  $k \in \mathbb{N}$ . In the context of this thesis, the assigned values by the labeling function are referred to as either labels or features, depending on the dimension of  $\Sigma$ . Additionally, the set of all graphs is denoted by  $\mathcal{G}$ .

Furthermore, a graph  $G$  can be either directed or undirected based on the definition of its set of edges  $E$ . If  $E \subseteq \{(v, u) \mid v, u \in V\}$ , it represents a directed graph, whereas if  $E \subseteq \{(v, u), (u, v) \mid v, u \in V, v \neq u\}$  such that for every  $(v, u) \in E$  there exists  $(u, v) \in E$ , it defines an undirected graph. Additionally, for ease of notation, we will use the  $V(G)$  and  $E(G)$  to denote the set of nodes and the set of edges of  $G$ , respectively, as well as  $l_G$  to denote the label function of  $G$ . With  $\mathcal{N}(v)$  for  $v \in V(G)$  we denote the set of neighbors of  $v$  defined as  $\mathcal{N}(v) := \{u \mid (u, v) \in E(G)\}$ .

**Definition 2** (Graph Coloring). A coloring of a Graph  $G$  is a function  $C : V(G) \rightarrow \mathbb{N}$  that assigns each node in the graph a color (here, a positive integer). Further, a coloring  $C$  induces a partition  $\mathcal{P}$  on the set of nodes, for which we define  $C^{-1}$  being the function that maps each color  $c \in \mathbb{N}$  to its class of nodes with  $C^{-1}(c) = \{v \in V(G) \mid C(v) = c\}$ . In addition, we define  $h_{G,C}$  as the histogram of graph  $G$  with coloring  $C$  that maps every color in the image of  $C$  under  $V(G)$  to the number of occurrences. In detail,  $\forall c \in \mathbb{N} : h_{G,C}(c) := |\{v \in V(G) \mid C(v) = c\}| = |C^{-1}(c)|$ .

#### Permutation-invariance and -equivariance

We use  $S_n$  to denote the symmetric group over the elements  $[n]$  for any  $n \in \mathbb{N}_{\geq 1}$ .  $S_n$  consists of all permutations over these elements. Let  $G$  be a graph with  $V(G) = [n]$ , applying a permutation  $\pi \in S_n$  on  $G$ , is defined as  $G_\pi := \pi \cdot G$  where  $V(G_\pi) = \{\pi(1), \dots, \pi(n)\}$  and  $E(G_\pi) = \{(\pi(v), \pi(u)) \mid (v, u) \in E(G)\}$ . We will now introduce two key concepts for classifying functions on graphs.

Changed  
[n] to be  
contain  
0

**Definition 3** (Permutation Invariant). Let  $f : \mathcal{G} \rightarrow \mathcal{X}$  be an arbitrary function, then  $f$  is *permutation-invariant* if and only if for all  $G \in \mathcal{G}$ , where  $n_G := |V(G)|$  and for every  $\pi \in S_{n_G}$ :  $f(G) = f(\pi \cdot G)$ .

**Definition 4** (Permutation Equivariant). Let  $f : \mathcal{G} \rightarrow \mathcal{X}$  be an arbitrary function, then  $f$  is *permutation-equivariant* if and only if for all  $G \in \mathcal{G}$ , where  $n_G := |V(G)|$  and for every  $\pi \in S_{n_G}$ :  $f(G) = \pi^{-1} \cdot f(\pi \cdot G)$ .

### 3.3. Weisfeiler and Leman Algorithm

The Weisfeiler-Leman algorithm consists of two main parts, first the coloring algorithm and second the graph isomorphism test. We will introduce them in this section.

#### The Weisfeiler-Leman Graph Coloring Algorithm

The 1-WL algorithm computes a node coloring of its input graph in each iteration. In detail, a color is assigned to each node based on the colors of its neighbors and its own current color. The algorithm continues until convergence is reached, resulting in the final coloring of the graph. We will now formally define this process and provide an illustrated example in Figure 2.

**Definition 5** (1-WL Algorithm). Let  $G = (V, E, l)$  be a labeled graph. In each iteration  $i$ , the 1-WL algorithm computes a node coloring  $C_i : V(G) \rightarrow \mathbb{N}$ . In the initial iteration  $i = 0$ , the coloring is set to  $C_0 = l$  if  $l$  exists. Otherwise, for all  $v \in V(G)$ ,  $C_0(v)$  is assigned an arbitrary constant value  $c \in \mathbb{N}$ . For  $i > 0$ , the algorithm assigns a color to  $v \in V(G)$  as follows:

$$C_i(v) = \text{RELABEL}(C_{i-1}(v), \{C_{i-1}(u) \mid u \in \mathcal{N}(v)\}),$$

where `RELABEL` injectively maps the above pair to a unique, previously not used, color. The algorithm terminates when the number of colors between two iterations does not change, meaning the algorithm terminates after iteration  $i$  if the following condition is satisfied:

$$\forall v, w \in V(G) : C_i(v) = C_i(w) \iff C_{i+1}(v) = C_{i+1}(w).$$

Upon terminating we define  $C_\infty := C_i$  as the stable coloring, such that  $1\text{-WL}(G) := C_\infty$ .

The colorings computed in each iteration always converge to the final one, such that the algorithm always terminates. In more detail, Grohe [2017] showed that it always holds after at most  $|V(G)|$  iterations. For an illustration of this algorithm, see Figure 2. Moreover, based on the work of Paige and Tarjan [1987] about efficient refinement strategies, Cardon and Crochemore [1982] proved that the stable coloring  $C_\infty$  can be computed in time  $\mathcal{O}(|V(G)| + |E(G)| \cdot \log |V(G)|)$ .

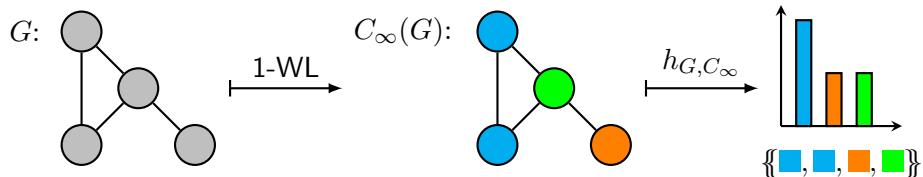


Figure 2.: An example of the final coloring computed by applying the 1-WL algorithm on the graph  $G$ . The graph  $G$  consists of 4 nodes with all their labels being set to the same color.

It is important to understand that since the algorithm was originally developed as a simple heuristic for the *graph isomorphism problem*, which is an inherently discrete problem, the 1-WL algorithm in its simplest form, as we presented it here, does only work on graphs with discrete, one-dimensional node labels. Although it is quite easy to adapt the algorithm to respect discrete edge labels of a graph by using them as weights in the neighborhood aggregation (Shervashidze et al. [2011]), modifying its definition to work with continuous graph features is more complex. Numerous proposed modifications have been put forward to address this integration in the literature, such as those discussed by Morris et al. [2016]. However, note that this particular topic will not be further investigated in this thesis, although its mention holds value for the following section.

### The Weisfeiler-Leman Graph Isomorphism Test

**Definition 6** (1-WL Isomorphism Test). To determine if two graphs  $G, H \in \mathcal{G}$  are non-isomorphic ( $G \not\cong H$ ), one applies the 1-WL coloring algorithm on both graphs “in parallel” and checks after each iteration if the occurrences of each color are equal, else the algorithm would terminate and conclude non-isomorphic. Formally, the algorithm concludes non-isomorphic in iteration  $i$  if there exists a color  $c$  such that:

$$|\{v \in V(G) \mid c = C_i(v)\}| \neq |\{v \in V(H) \mid c = C_i(v)\}|.$$

Note that this test is only sound and not complete for the *graph isomorphism problem*. Counterexamples can be easily constructed where the algorithm fails to distinguish non-isomorphic graphs. See Figure 3, discovered and proven by Cai et al. [1992].

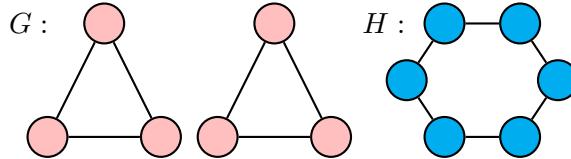


Figure 3.: This is an example of two graphs  $G$  and  $H$  that are non-isomorphic but cannot be distinguished by the 1-WL isomorphism test.

### Implications of the Weisfeiler-Leman Algorithm

In this section, we introduce two basic concepts: the 1-WL relation and the notion of 1-WL-Discriminating property for permutation invariant functions.

**Definition 7** (1-WL Relation). For any graphs  $G, H \in \mathcal{X}$  we will denote  $G \simeq_{1WL} H$  if the 1-WL isomorphism test can not distinguish both graphs. Note that due to the soundness of this algorithm, if  $G \not\simeq_{1WL} H$ , we always can conclude that  $G \not\cong H$ .

The  $\simeq_{1WL}$  relation can further be classified as an equivalence relation, as it is reflexive, symmetric and transitive. With this, we introduce a notation of its equivalence classes. Let  $\mathcal{X} \subseteq \mathcal{G}$  and  $G \in \mathcal{X}$ , then we denote with  $\mathcal{X}/\simeq_{1WL}(G) := \{G' \in \mathcal{X} \mid G \simeq_{1WL} G'\}$  its equivalence class.

**Definition 8** (1-WL-Discriminating). Let  $\mathcal{C}$  be a collection of permutation invariant functions from  $\mathcal{X}$  to  $\mathbb{R}$ . We say  $\mathcal{C}$  is **1-WL-Discriminating** if for all graphs  $G_1, G_2 \in \mathcal{X}$  for which the 1-WL isomorphism test concludes non-isomorphic ( $G_1 \not\simeq_{1WL} G_2$ ), there exists a function  $h \in \mathcal{C}$  such that  $h(G_1) \neq h(G_2)$ .

### 3.4. 1-WL+NN

As the previous section shows, the 1-WL algorithm is quite powerful in identifying a graph's substructures and distinguishing non-isomorphic graph pairs. With the 1-WL+NN framework, we define functions that utilize this structural information to derive further application-specific insights.

**Definition 9** (1-WL+NN). A 1-WL+NN model consists of three components that are applied sequentially to its input: 1. the 1-WL algorithm, 2. an encoding function  $f_{\text{enc}}$  operating on multisets of  $\mathbb{N}$ , and 3. an arbitrary multilayer perceptron MLP. In detail, the model computes the function  $\mathcal{B}$  as follows:

$$\mathcal{B} : \mathcal{G} \rightarrow \mathbb{R}^k, G \mapsto \text{MLP} \circ f_{\text{enc}}(\{\{1\text{-WL}(G)(v) \mid v \in V(G)\}\}),$$

where “1-WL( $G$ )” is the coloring computed by the 1-WL algorithm when applied on  $G$ , and  $k \in \mathbb{N}$  is an arbitrary constant. For a better understanding and an illustrative explanation, see Figure 4.

It is worth noting that this definition can be easily adjusted to accommodate node or edge-related tasks by applying the encoding function  $f_{\text{enc}}$  and the multilayer perceptron MLP elementwise to the colors of the multiset. However, for the purposes of this thesis, we will not delve into these variations, as our main focus will be on graph-wide tasks such as graph classification or regression, which possess greater theoretical interest and are more prevalent in most datasets. Furthermore, all the theoretical findings presented in this thesis can also be applied to 1-WL+NN models designed for node or edge tasks.

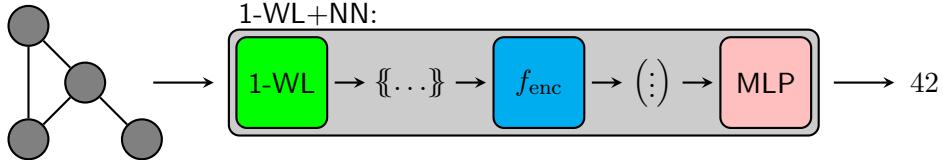


Figure 4.: This simplified illustration explains the components that make up a 1-WL+NN model and how each one processes the input. In detail, the model takes the graph on the left as input and first applies the 1-WL algorithm, thereby obtaining a multiset of the colors assigned by the algorithm. Then the encoding function  $f_{\text{enc}}$  is applied, resulting in a fixed-size vector further processed by the multilayer perceptron MLP. The output of the MLP is then propagated as the 1-WL+NN models output, here the number 42.

### 3.5. Graph Neural Networks (Message Passing)

A Graph Neural Network (GNN) is a composition of multiple layers, where each layer computes a new feature for each node and edge. Each GNN layer thus technically obtains a new graph structurally identical to the previous one but contains new feature information. After an input graph has been passed through all layers, a final readout function is applied that pools all graph features and derives a task-related output. With this, it is possible to apply a GNN to every graph, regardless of its size, as the “computation” will only take place on the nodes and edges of the graph.

Note that in the following, we will restrict the definition only to consider node features; however, one can easily extend it to include edge features as well.

**Definition 10** (Graph Neural Network). Let  $G = (V, E, l)$  be an arbitrary graph. A Graph Neural Network (GNN) is a composition of multiple layers and a final readout function where each layer  $t$  is represented by a function  $f^{(t)}$ . The initial layer at  $t = 0$  is a functioning of the format  $f^{(0)} : V(G) \rightarrow \mathbb{R}^{1 \times d}$  that is consistent with  $l$  and translates all labels into a vector representation. In contrast, for every  $t > 0$ ,  $f^{(t)}$  is recursively defined as:

$$f^{(t)}(v) = f_{\text{merge}}^{(t)}(f^{(t-1)}(v), f_{\text{agg}}^{(t)}(\{f^{(t-1)}(w) \mid w \in \mathcal{N}(v)\})),$$

where  $f_{\text{merge}}^{(t)}$  is an arbitrary function that maps the aforementioned tuple to a vector, effectively “merging” them, while  $f_{\text{agg}}^{(t)}$  is an arbitrary function that maps the multiset to a vector, effectively “aggregating” it.

The readout function, referred to as **Readout**, is applied after the input graph has been passed subsequently through all layers and is defined as follows:

$$\text{Readout}(\{f^{(t)}(v) \mid v \in V(G)\}).$$

This function pools the information from every node feature, processes it, and calculates a fixed-size output vector for the entire graph.

In summary, a GNN model will compute the function  $\mathcal{A}$  as follows:

$$\mathcal{A} : \mathcal{G} \rightarrow \mathbb{R}^k, G \mapsto \text{Readout}(\{f^{(T)}(v) \mid v \in V(G)\}),$$

where  $T$  is the number of layer of the GNN, and  $k \in \mathbb{N}$  an arbitrary constant. To enable end-to-end training of a GNN, it is essential that all its components are differentiable. Therefore, we require all  $f^{(t)}$  merge and  $f^{(t)}$  agg functions, along with the final **Readout** function, to be differentiable.

Note that, due to our definition of the “aggregation” function and the “readout” function to operate over multisets, both functions are permutation invariant by definition. With this, we can conclude that the total composition  $\mathcal{A}$  is permutation-invariant, and with similar reasoning, it is also differentiable. This property enables us to train  $\mathcal{A}$  like any other machine learning method in an end-to-end fashion, regardless of the underlying encoding used for graphs. The definition and notation used here are inspired by Morris et al. [2019] and Xu et al. [2019].

To bridge the gap from the theoretical definition to practical instance of the definition, we will now introduce three distinct GNNs. Specifically, we will explore the Graph Convolutional Network (GCN) proposed by Kipf and Welling [2017], the Graph Isomorphism Network (GIN) introduced by Xu et al. [2019], and the Graph Attention Network (GAT) presented by Veličković et al. [2017]. These GNNs will serve as empirical baselines in Part II. The update equations for each GNN layer, denoted as  $f^{(t)}$  in the definition, are listed in Table 1.

Commonly employed readout functions in this context often involve straightforward pooling techniques like elementwise summation, mean calculation, or maximum extraction. These pooling operations are typically followed by a multilayer perceptron, which performs additional processing on the aggregated information.

Common readout functions for GNNs, often involve simple pooling techniques like elementwise summation, averaging or maximum extraction. These pooling operations are typically followed by a multilayer perceptron, which performs additional processing on the aggregated information.

Finish  
GNNs

Model	Update Equation
GCN	$f^{(t)}(v) = \text{ReLU} \left( \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{W^{(t)}}{\sqrt{(1+d(v))(1+d(u))}} f^{(t-1)}(u) \right)$
GIN	$f^{(t)}(v) = \text{MLP}^{(k)} \left( (1 + \epsilon^{(k)}) f^{(t-1)}(v) + \sum_{u \in \mathcal{N}(v)} f^{(t-1)}(u) \right)$
GAT	$f^{(t)}(v) = \sigma \left( \sum_{u \in \mathcal{N}(v)} \alpha_{vu} W^{(t)} f^{(t-1)}(u) \right)$

Table 1.: Overview of the update equations used by popular GNN configuration. This format is inspired by Nikolentzos et al. [2023a].

## 4. Theoretical Connection

This section is the main part of our theoretical investigation of the two frameworks. We will present four intriguing theorems, which will be proven separately afterward. In detail, the first two theorems will establish an equivalence between the two frameworks when the input set of graphs is finite. In comparison, the last two theorems will go one step further and establish a connection for continuous functions computed by 1-WL+NN and GNNs and prove a somewhat weaker connection between them.

In the first two theorems, we focus on a finite collection of graphs, which we denote by  $\mathcal{X} \subset \mathcal{G}$ .

**Theorem 11** (Finite Case: “GNN  $\subseteq$  1-WL+NN”). Let  $\mathcal{C}$  be a collection of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by GNNs, then  $\mathcal{C}$  is also computable by 1-WL+NN.

**Theorem 12** (Finite Case: “1-WL+NN  $\subseteq$  GNN”). Let  $\mathcal{C}$  be a collection of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by 1-WL+NN, then  $\mathcal{C}$  is also computable by GNNs.

With these two theorems, we showed the equivalence between both frameworks. Specifically, every function computed by 1-WL+NN working over any arbitrary  $\mathcal{X} \subset \mathcal{G}$  is also computable by a GNN, and vice versa. Notice that we did not leverage any constraints on the encoding of graphs throughout the first two theorems and their corresponding proves but instead kept it general.

Having established a connection between the two frameworks on a finite subset of graphs, we wanted to further demonstrate the expressive power of 1-WL-Discriminating by investigating a connection between the two frameworks for continuous feature spaces and continuous functions. However, since the *graph isomorphism problem* is an inherently discrete problem, the 1-WL algorithm, as outlined in Section 3.4, is only defined as a discrete and discontinuous function operating on discrete colors, such that extending the definition of the 1-WL algorithm to a continuous function working over continuous values is not very trivial and, to our knowledge, has not yet been widely investigated. Therefore, we assume in the proofs and the following theorems that a continuous version of the 1-WL algorithm exists.

We define the set of graphs with continuous features using the following definition:

**Definition 13.** Let  $X$  be a compact subset of  $\mathbb{R}$  including 0. We decode graphs with  $n$  nodes as a matrix  $G \in X^{n \times n}$ , where  $G_{i,i}$  decodes the label of node  $i$  for  $i \in [n]$ , and  $G_{i,j}$  with  $i \neq j \in [n]$  decodes an edge from node  $i$  to  $j$  and a corresponding edge features. Furthermore, we say that there is an edge between node  $i$  and  $j$  if and only if  $G_{i,j} \neq 0$ . Additionally, if  $G$  encodes an

undirected graph,  $G$  is a symmetric matrix. For simplicity, we denote  $\mathcal{X} := X^{n \times n}$  throughout the next two theorems and their respective proofs.

**Theorem 14** (Continuous Case: “GNN  $\subseteq$  1-WL+NN”). Let  $\mathcal{C}$  be a collection of continuous functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by 1-WL+NN. If  $\mathcal{C}$  is 1-WL-Discriminating, then there exists a collection of functions  $\mathcal{C}'$  computable by 1-WL+NN that is GNN-Approximating.

**Theorem 15** (Continuous Case: “1-WL+NN  $\subseteq$  GNN”). Let  $\mathcal{C}$  be a collection of continuous functions from  $\mathcal{X}$  to  $\mathbb{R}$  that is GNN-Approximating, then  $\mathcal{C}$  is also 1-WL-Discriminating.

Immediately from the last theorem follows the corollary:

**Corollary 16.** There exists a collection  $\mathcal{C}$  of function from  $\mathcal{X}$  to  $\mathbb{R}$  computable by GNNs that is 1-WL-Discriminating.

*Proof.* The collection of all functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by GNNs is trivially GNN-Approximating, such that we can apply Theorem 15 with which the proof concludes.  $\square$

Since we only wanted to show the expressive power of 1-WL-Discriminating and made the major assumption of the existence of a continuous 1-WL algorithm, we have included the proofs of Theorems 14 and 15 in the Appendix in ??.

By putting both theorems into perspective, we can now argue that even for continuous functions 1-WL+NN and GNNs can compute almost the same functions. Each framework can approximate the other framework arbitrarily well. We can conclude that the ability 1-WL-Discriminating is very powerful, so we can assume that 1-WL+NN is sufficiently expressive for the upcoming empirical part.

#### 4.1. Proof of Theorem 11

We will prove Theorem 11 by introducing a couple of small lemmas, which combined prove the theorem. In detail, in Lemma 17, we show the existence of a collection computed by 1-WL+NN that is 1-WL-Discriminating. In Lemmas 18 to 20 we derive properties of 1-WL+NN functions we will use throughout Lemmas 21 to 23 with which we prove the theorem. We took great inspiration for Lemmas 21 to 23 from the proof presented in section 3.1 in the work of Chen et al. [2019].

**Lemma 17.** There exists a collection  $\mathcal{C}$  of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by 1-WL+NN that is 1-WL-Discriminating.

*Proof.* We define  $f_c$  for  $c \in \mathbb{N}$  as the encoding function that returns the number of nodes colored as  $c$ . With this, we can construct the collection of functions  $C$  as follows:

$$C := \{\mathcal{B}_c : \mathcal{X} \rightarrow \mathbb{R}, G \mapsto \text{MLP}_{\text{id}} \circ f_c(\{1\text{-WL}(G)(v) \mid v \in V(G)\}) \mid c \in \mathbb{N}\},$$

where  $\text{MLP}_{\text{id}}$  is a dummy multilayer perceptron that returns its input. Since every function  $\mathcal{B}_c \in C$  is composed of the 1-WL algorithm, an encoding function, and a multilayer perceptron, each function is computable by 1-WL+NN, and consequently also the whole collection.

Let  $G_1, G_2 \in \mathcal{X}$  with  $G_1 \not\approx_{1\text{-WL}} G_2$ . Further, let  $C_1, C_2$  be the final colorings computed by the 1-WL algorithm when applied on  $G_1, G_2$  respectively. Due to  $G_1 \not\approx_{1\text{-WL}} G_2$ , there exists a color  $c \in \mathbb{N}$  such that  $h_{G_1, C_1}(c) \neq h_{G_2, C_2}(c)$ . Such that  $\mathcal{B}_c \in C$  exists with  $\mathcal{B}_c(G_1) \neq \mathcal{B}_c(G_2)$ .  $\square$

**Lemma 18** (1-WL+NN Equivalence). Let  $\mathcal{B}$  be a function over  $\mathcal{X}$  computable by 1-WL+NN, then for every pair of graphs  $G_1, G_2 \in \mathcal{X}$  : if  $G_1 \simeq_{1WL} G_2$  than  $\mathcal{B}(G_1) = \mathcal{B}(G_2)$ .

*Proof.* Let  $\mathcal{B}$  be an arbitrary function over  $\mathcal{X}$  computable by 1-WL+NN, then  $\mathcal{B}$  is composed as follows:  $\mathcal{B}(\cdot) = \text{MLP} \circ f_{\text{enc}} \{\{1\text{-WL}(\cdot)(v) \mid v \in V(G)\}\}$ . Further, let  $G_1, G_2 \in \mathcal{X}$  be arbitrary graphs with  $G_1 \simeq_{1WL} G_2$ , then by definition of the relation  $\simeq_{1WL}$  we know that  $1\text{-WL}(G_1) = 1\text{-WL}(G_2)$ . With this, the equivalence follows immediately.  $\square$

**Lemma 19** (1-WL+NN Permutation Invariance). Let  $\mathcal{B}$  be a function over  $\mathcal{X}$  computable by 1-WL+NN, then  $\mathcal{B}$  is permutation invariant.

*Proof.* Let  $G_1, G_2 \in \mathcal{X}$  be arbitrary graphs with  $G_1 \simeq G_2$  and  $\mathcal{B}$  an arbitrary function computable by 1-WL+NN. Since the 1-WL algorithm is sound, we know that  $G_1 \simeq G_2$  implies  $G_1 \simeq_{1WL} G_2$ . Using Lemma 18, we can therefore conclude that:  $\mathcal{B}(G_1) = \mathcal{B}(G_2)$ .  $\square$

**Lemma 20** (1-WL+NN Composition). Let  $\mathcal{C}$  be a collection of functions computable by 1-WL+NN. Further, let  $h_1, \dots, h_n \in \mathcal{C}$  and  $\text{MLP}^*$  an multilayer perceptron, than the function  $\mathcal{B}$  composed of  $\mathcal{B}(\cdot) := \text{MLP}^*(h_1(\cdot), \dots, h_n(\cdot))$  is also computable by 1-WL+NN.

*Proof Sketch.* Assume the above and let  $f_1, \dots, f_n$  be the encoding functions, as well as  $\text{MLP}_1, \dots, \text{MLP}_n$  be the multilayer perceptrons used by  $h_1, \dots, h_n$  respectively. The idea of this proof is that we construct an encoding function  $f^*$  that “duplicates” its input and applies each encoding function  $f_i$  individually. We also construct a multilayer perceptron  $\text{MLP}^*$  that takes in the output of  $f^*$  and simulates all  $\text{MLP}_1, \dots, \text{MLP}_n$  simultaneously. Afterward, the given  $\text{MLP}^*$  will be applied on the concatenation of the output of all  $mlp_i$ s. See Figure 5 for a sketch of the proof idea. For the complete proof, please refer to the Appendix in subsection 2.1.

$$G \xrightarrow{1\text{-WL}} \{1 - WL(G)(v) \mid v \in V(G)\} \xrightarrow{f^*} \begin{bmatrix} f_1(M_G) \\ \vdots \\ f_n(M_G) \end{bmatrix} \xrightarrow{\text{MLP}^*} \text{MLP}^*\left(\begin{bmatrix} \text{MLP}_1(f_1(M_G)) \\ \vdots \\ \text{MLP}_n(f_n(M_G)) \end{bmatrix}\right)$$

Figure 5.: The proof idea for Lemma 20, how the constructed functions  $f^*$  and  $\text{MLP}^*$  will work on input  $G \in \mathcal{X}$ . Here we denote with  $M_G$  the multiset of colors of the nodes of  $G$  after applying the 1-WL algorithm.

$\square$

**Lemma 21.** Let  $\mathcal{C}$  be a collection of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by 1-WL+NN that is 1-WL-Discriminating. Then for all  $G^* \in \mathcal{X}$ , there exists a function  $h_{G^*}$  from  $\mathcal{X}$  to  $\mathbb{R}$  computable by 1-WL+NN, such that for all  $G \in \mathcal{X}$ :  $h_{G^*}(G) = 0$ , if and only if,  $G \simeq_{1WL} G^*$ .

*Proof.* Assume the above. For any  $G_1, G_2 \in \mathcal{X}$  with  $G_1 \not\simeq_{1WL} G_2$ , let  $h_{G_1, G_2} \in \mathcal{C}$  be the function distinguishing them, with  $h_{G_1, G_2}(G_1) \neq h_{G_1, G_2}(G_2)$ . We define the function  $\bar{h}_{G_1, G_2}$  working over  $\mathcal{X}$  as follows:

$$\begin{aligned} \bar{h}_{G_1, G_2}(\cdot) &= |h_{G_1, G_2}(\cdot) - h_{G_1, G_2}(G_1)| \\ &= \max(h_{G_1, G_2}(\cdot) - h_{G_1, G_2}(G_1), 0) + \max(h_{G_1, G_2}(G_1) - h_{G_1, G_2}(\cdot), 0) \\ &= \text{ReLU}(h_{G_1, G_2}(\cdot) - h_{G_1, G_2}(G_1)) + \text{ReLU}(h_{G_1, G_2}(G_1) - h_{G_1, G_2}(\cdot)) \end{aligned} \quad (0.1)$$

Note, that in the equations above “ $h_{G_1, G_2}(G_1)$ ” is a fixed constant and the resulting function  $\bar{h}_{G_1, G_2}$  is non-negative. Let  $G_1 \in \mathcal{X}$  now be fixed, we will construct the function  $h_{G_1}$  with the desired properties as follows:

$$h_{G_1}(\cdot) = \sum_{G_2 \in \mathcal{X}, G_1 \not\simeq_{\text{WL}} G_2} \bar{h}_{G_1, G_2}(\cdot). \quad (0.2)$$

Since  $\mathcal{X}$  is finite, the sum is finite and therefore well-defined. Next, we will prove that for a fixed graph  $G_1 \in \mathcal{X}$ , the function  $h_{G_1}$  is correct on input  $G \in \mathcal{X}$ :

1. If  $G_1 \simeq_{\text{WL}} G$ , then for every function  $\bar{h}_{G_1, G_2}$  of the sum with  $G_1 \not\simeq_{\text{WL}} G_2$ , we know, using Lemma 18, that  $\bar{h}_{G_1, G_2}(G)$  is equal to  $\bar{h}_{G_1, G_2}(G_1)$  which is by definition 0, such that  $h_{G_1}(G) = 0$ .
2. If  $G_1 \not\simeq_{\text{WL}} G$ , then  $\bar{h}_{G_1, G}(G)$  is a summand of the overall sum, and since  $\bar{h}_{G_1, G}(G) > 0$ , we can conclude  $h_{G_1}(G) > 0$  due to the non-negativity of each function  $\bar{h}_{G_1, G_2}$ .

Using Lemma 20, we can conclude that for any  $G \in \mathcal{X}$ ,  $h_G$  is computable by 1-WL+NN, as we can encode Equation 0.2 via a multilayer perceptron where the factor “ $h_{G_1, G_2}(G_1)$ ” of Equation 0.1 is just a constant.  $\square$

**Lemma 22.** Let  $\mathcal{C}$  be a collection of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by 1-WL+NN so that for all  $G^* \in \mathcal{X}$ , there exists  $h_{G^*} \in \mathcal{C}$  satisfying  $h_{G^*}(G) = 0$  if and only if  $G \simeq_{\text{WL}} G^*$  for all  $G \in \mathcal{X}$ . Then for every  $G^* \in \mathcal{X}$ , there exists a function  $\varphi_{G^*}$  computable by 1-WL+NN such that for all  $G \in \mathcal{X}$ :  $\varphi_{G^*}(G) = \mathbf{1}_{G \simeq_{\text{WL}} G^*}$ .

*Proof.* Assuming the above. Due to  $\mathcal{X}$  being finite, we can define for every graph  $G^*$  the constant:

$$\delta_{G^*} := \frac{1}{2} \min_{G \in \mathcal{X}, G \not\simeq_{\text{WL}} G^*} |h_{G^*}(G)| > 0.$$

With this constant, we can use a so-called “bump” function working from  $\mathbb{R}$  to  $\mathbb{R}$  that is similar to the indicator function. We define this function for parameter  $a \in \mathbb{R}$  with  $a > 0$  as:

$$\begin{aligned} \psi_a(x) &:= \max\left(\frac{x}{a} - 1, 0\right) + \max\left(\frac{x}{a} + 1, 0\right) - 2 \cdot \max\left(\frac{x}{a}, 0\right) \\ &= \text{ReLU}\left(\frac{x}{a} - 1\right) + \text{ReLU}\left(\frac{x}{a} + 1\right) - 2 \cdot \text{ReLU}\left(\frac{x}{a}\right) \end{aligned} \quad (0.3)$$

The interesting property of  $\psi_a$  is that it maps every value  $x$  to 0, except when  $x$  is being drawn from the interval  $(-a, a)$ . In particular, it maps  $x$  to 1 if and only if  $x$  is equal to 0. See Figure 6 for a plot of the relevant part of this function with exemplary values for  $a$ .

We use these properties to define for every graph  $G^* \in \mathcal{X}$  the function  $\varphi_{G^*}(\cdot) := \psi_{\delta_{G^*}}(h_{G^*}(\cdot))$ . We will quickly demonstrate that this function is equal to the indicator function, for this let  $G^*$  be fixed and  $G$ , an arbitrary graph from  $\mathcal{X}$ , the input:

1. If  $G \simeq_{\text{WL}} G^*$ , then  $h_{G^*}(G) = 0$  resulting in  $\varphi_{G^*}(G) = \psi_{\delta_{G^*}}(0) = 1$ .
2. If  $G \not\simeq_{\text{WL}} G^*$  then  $h_{G^*}(G) \neq 0$ , such that  $|h_{G^*}(G)| > \delta_{G^*}$  so that  $h_{G^*}(G) \notin (-\delta_{G^*}, \delta_{G^*})$  resulting in  $\varphi_{G^*}(G) = 0$ .

Note that we can encode  $\varphi_{G^*}$  using Equation 0.3 via a multilayer perceptron, where  $\delta_{G^*}$  is a constant. With Lemma 20 we can therefore conclude that  $\varphi_{G^*}$  is computable by 1-WL+NN for every graph  $G^* \in \mathcal{X}$ .  $\square$

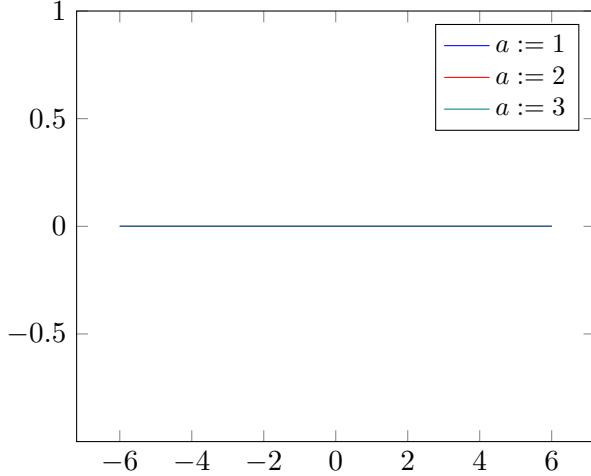


Figure 6.: Illustration of the so-called “bump” function  $\psi_a(x)$  used in the proof of Lemma 22 with different exemplary values for  $a$ .

**Lemma 23.** Let  $\mathcal{C}$  be a collection of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by 1-WL+NN such that for all  $G^* \in \mathcal{X}$ , there exists  $\varphi_{G^*} \in \mathcal{C}$  satisfying  $\forall G \in \mathcal{X} : \varphi_{G^*}(G) = \mathbf{1}_{G \simeq_{1\text{-WL}} G^*}$ . Then every function computable by a GNN is also computable by 1-WL+NN.

*Proof.* Assume the above. For any function  $\mathcal{A}$  computed by a GNN that works over  $\mathcal{X}$  to  $\mathbb{R}$ , we show that it can be decomposed as follows for any  $G \in \mathcal{X}$  as input:

$$\begin{aligned} \mathcal{A}(G) &= \left( \frac{1}{|\mathcal{X}/\simeq_{1\text{-WL}}(G)|} \sum_{G^* \in \mathcal{X}} \mathbf{1}_{G \simeq_{1\text{-WL}} G^*} \right) \cdot \mathcal{A}(G) \\ &= \frac{1}{|\mathcal{X}/\simeq_{1\text{-WL}}(G)|} \sum_{G^* \in \mathcal{X}} \mathcal{A}(G^*) \cdot \mathbf{1}_{G \simeq_{1\text{-WL}} G^*} \\ &= \sum_{G^* \in \mathcal{X}} \frac{\mathcal{A}(G^*)}{|\mathcal{X}/\simeq_{1\text{-WL}}(G^*)|} \cdot \varphi_{G^*}(G) \end{aligned} \quad (0.4)$$

where we denote with  $\mathcal{X}/\simeq_{1\text{-WL}}(G^*)$  the set of all graphs  $G$  over  $\mathcal{X}$  that are equivalent to  $G^*$  according to the  $\simeq_{1\text{-WL}}$  relation.

Since  $\mathcal{A}$  is permutation-invariant and GNNs are, at most, as good as the 1-WL algorithm in distinguishing non-isomorphic graphs, we can use the fact that for every pair of graphs  $G, H \in \mathcal{X}$  with  $G \simeq_{1\text{-WL}} H$ :  $\mathcal{A}(G) = \mathcal{A}(H)$ . Therefore, we can decompose  $\mathcal{A}$  as indicated in Equation 0.4 and encode it using a multilayer perceptron where  $\frac{\mathcal{A}(G^*)}{|\mathcal{X}/\simeq_{1\text{-WL}}(G^*)|}$  is a constant, and  $\varphi_{G^*} \in \mathcal{C}$  encodes the indicator function. Combined with the Lemma 20, we can conclude that  $\mathcal{A}$  is computable by 1-WL+NN. Important to note, we can only do this since  $\mathcal{X}$  is finite, making the overall sum finite and the cardinality of  $\mathcal{X}/\simeq_{1\text{-WL}}(G^*)$  well-defined for all graphs.  $\square$

## 4.2. Proof of Theorem 12

In this section we will prove the converse direction. We start with Lemma 24, where we introduce an upper bound that we will use in Lemma 25 to show that there exists a collection of GNN-computable functions that is 1-WL-Discriminating. After that, we will prove a composition lemma with Lemma 26 which is similar to the one we introduced in the previous section. From

this point on, the proof continues as in the previous section and concludes the property to be proved in Lemma 27.

**Lemma 24.** Let  $G$  be an arbitrary graph with  $n := |V(G)|$  the number of nodes and  $C : V(G) \rightarrow \mathbb{N}$  an arbitrary coloring of the graph  $G$ . Then the total number of possible tuples of the form:

$$(C(v), \{\{C(u) \mid u \in \mathcal{N}(v)\}\}),$$

for all  $v \in V(G)$  can be upper bounded by:

$$n \cdot \sum_{i=0}^{n-1} \binom{n+i-1}{i}.$$

*Proof.* Assume the above. For the first entry of the tuple, at most  $n$  different colors exist since there are  $n$  nodes. For the second entry, each node  $v \in V(G)$  can have between 0 and  $n-1$  neighbors, such that the total number of possibilities is the sum over each cardinality of a multiset with  $n$  different colors. In the end, we soundly combine both results by multiplying both together.  $\square$

**Lemma 25** (GNN 1-WL-Discriminating). There exists a collection  $\mathcal{C}$  of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by GNNs that is 1-WL-Discriminating. Meaning for every  $G_1, G_2 \in \mathcal{X}$  with  $G_1 \not\leq_{1WL} G_2$  there exists  $\mathcal{A} \in \mathcal{C}$  such that  $\mathcal{A}(G_1) \neq \mathcal{A}(G_2)$ .

*Proof.* Since  $\mathcal{X}$  is finite, we define  $n := \max\{|V(G)| \mid G \in \mathcal{X}\}$  to be the maximum number of nodes of a graph in  $\mathcal{X}$ , and  $k := \max\{l_G(v) \mid v \in V(G), G \in \mathcal{X}\}$  to be the largest label of any node of a graph in  $\mathcal{X}$ . Using Lemma 24, we can compute an upper bound  $m$  using  $n$  for the number of distinct tuples. Note that, this bound holds true for all graphs in  $\mathcal{X}$ . We will now construct a GNN with  $n$  layers working on input  $G$  as follows:

$$\begin{aligned} f^{(0)}(v) &:= l_G(v), \text{ and} \\ f^{(t)}(v) &:= \text{RELABEL}_{m,t}(f^{(t-1)}(v), \{\{f^{(t-1)}(u) \mid u \in \mathcal{N}(v)\}\}), \quad 0 < t < n. \end{aligned}$$

Here  $\text{RELABEL}_{m,t}$  is a function that maps the tuples injectively to an integer of the set:

$$\{i \in \mathbb{N} \mid k + (t-1) \cdot m + 1 \leq i \leq k + t \cdot m\}.$$

This function exists as by the soundness of the upper bound of Lemma 24, the cardinality of its co-domain is greater or equal than the one of its domain. Thereby, and with the injectiveness of  $\text{RELABEL}_{m,t}$ , we ensure that each GNN layer maps a tuple to a new, previously unused color. Therefore, every layer of this GNN computes a single iteration of the 1-WL algorithm. Further, since the 1-WL algorithm converges after at most  $|V(G)| \leq n$  iterations, we set the number of layers to  $n$ . We ensure that the coloring computed by this GNN after  $n$  layers when applied on any graph  $G \in \mathcal{X}$  is similarly expressive as the coloring computed by the 1-WL algorithm when applied on  $G$ .

We define the collection  $\mathcal{C}$  of functions computable by GNNs that is 1-WL-Discriminating as:

$$\mathcal{C} := \{\mathcal{A} : \mathcal{X} \rightarrow \mathbb{R}, G \mapsto \text{Readout}_c(\{f^{(n)}(v) \mid v \in V(G)\}) \mid c \in \mathbb{N}\},$$

where  $\text{Readout}_c$  is the Readout function that returns the number of nodes colored as  $c$  in the coloring of  $f^{(n)}$ .  $\square$

Similar to the proof in the previous section, we will use Lemma 26 to introduce the ability to construct GNNs that take in as input multiple GNNs and then apply a multilayer perceptron to the combined output.

**Lemma 26** (Composition GNN). Let  $C$  be a collection of function computable by GNNs. Further, let  $\mathcal{A}_1, \dots, \mathcal{A}_n \in C$  and  $\text{MLP}^\bullet$  a suitable multilayer perceptron, then  $\hat{\mathcal{A}}(\cdot) := \text{MLP}(\mathcal{A}_1(\cdot), \dots, \mathcal{A}_n(\cdot))$  is also computable by a GNN.

*Proof.* Further, for any  $x \in \mathbb{R}^d$ , we will use the notation  $x[i]$  to indicate the  $i$ .th element of the vector  $x$ . Additionally, we indicate the merge and aggregation function used in layer  $t$  by  $\mathcal{A}_i$  as  $f_{\text{merge},i}^{(t)}$  and  $f_{\text{agg},i}^{(t)}$ . Similarly, does  $\text{Readout}_i$  indicate the  $\text{Readout}$  function and  $f_i^{(0)}$  the input function of  $\mathcal{A}_i$ .

We will prove the lemma by constructing  $\hat{\mathcal{A}}$ . Let  $T$  be the maximum number of layers of all  $\mathcal{A}_1, \dots, \mathcal{A}_n$ . We construct the GNN  $\hat{\mathcal{A}}$  with  $T$  layers, with the input layer working as follows on an input graph  $G$ :

$$\forall v \in V(G) : \hat{f}^{(0)}(v) := \begin{bmatrix} f_1^{(0)}(v) \\ \vdots \\ f_n^{(0)}(v) \end{bmatrix},$$

and each other layer  $0 < t \leq K$  uses the merge  $\hat{f}_{\text{merge}}^{(t)}$  and aggregation  $\hat{f}_{\text{agg}}^{(t)}$  functions as defined below:

$$\begin{aligned} \hat{f}_{\text{merge}}^{(t)}(\hat{f}^{(t-1)}(v), \text{Agg}) &:= \begin{bmatrix} f_{\text{merge},1}^{(t)}(\hat{f}^{(t-1)}(v)[1], \text{Agg}[1]) \\ \vdots \\ f_{\text{merge},n}^{(t)}(\hat{f}^{(t-1)}(v)[n], \text{Agg}[n]) \end{bmatrix}, \quad \text{and} \\ \hat{f}_{\text{agg}}^{(t)}(\{\hat{f}^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}) &:= \begin{bmatrix} f_{\text{agg},1}^{(t)}(\{f^{(t-1)}(w)[1] \mid w \in \mathcal{N}(v)\}) \\ \vdots \\ f_{\text{agg},n}^{(t)}(\{f^{(t-1)}(w)[n] \mid w \in \mathcal{N}(v)\}) \end{bmatrix}. \end{aligned}$$

Note that, not all  $\mathcal{A}_i$  will be comprised of  $K$  layers. For these cases we define the missing functions as follows:

$$\begin{aligned} f_{\text{merge},i}^{(t)}(\hat{f}^{(t-1)}(v), \text{Agg}) &:= \hat{f}^{(t-1)}(v), \quad \text{and} \\ f_{\text{agg},i}^{(t)}(\{f^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}) &:= 0. \end{aligned}$$

These functions do not change anything, and only forward the result of the actual computation of  $\mathcal{A}_i$  to the last layer. Finally, we construct the  $\text{Readout}$  function of  $\hat{\mathcal{A}}$  as follows:

$$\text{Readout}(\{\hat{f}^{(T)}(v) \mid v \in V(G)\}) := \text{MLP} \circ \begin{bmatrix} \text{Readout}_1(\{\hat{f}^{(T)}(v)[1] \mid v \in V(G)\}) \\ \vdots \\ \text{Readout}_n(\{\hat{f}^{(T)}(v)[n] \mid v \in V(G)\}) \end{bmatrix}.$$

□

As a consequence of the previous two lemmas, we find ourselves in a similar position as at the beginning of the proof in Section 4.1. Specifically, we have established, through Lemma 25, the existence of a collection  $C$  of functions that can be computed by GNNs and can effectively distinguish any pair of graphs that are also distinguishable by the 1-WL algorithm. Furthermore, with Lemma 26, we have demonstrated that the composition of multiple GNNs and a multilayer perceptron remains computable by a single GNN. Consequently, we can also apply the findings of Lemmas 21 and 22 to GNNs. Thus, we can conclude that for any fixed  $G^* \in \mathcal{X}$ , the indicator function  $\varphi_{G^*}$  working over  $\mathcal{X}$  with:

$$\forall G \in \mathcal{X} : \quad \varphi_{G^*}(G) := \begin{cases} 1, & \text{if } G \simeq_{1\text{WL}} G^* \\ 0, & \text{else} \end{cases},$$

is computable by a GNN.

**Lemma 27.** Let  $\mathcal{C}$  be a collection of functions from  $\mathcal{X}$  to  $\mathbb{R}$  computable by GNNs so that for all  $G^* \in \mathcal{X}$ , there exists  $\varphi_{G^*} \in \mathcal{C}$  satisfying  $\forall G \in \mathcal{X} : \varphi_{G^*}(G) = \mathbb{1}_{G \simeq_{1\text{WL}} G^*}$ . Then every function computable by 1-WL+NN is also computable by a GNN.

*Proof.* Assume the above. For any function  $\mathcal{B}$  computed by 1-WL+NN that works over  $\mathcal{X}$  to  $\mathbb{R}$ , we show that it can be decomposed as follows for any  $G \in \mathcal{X}$  as input:

$$\begin{aligned} \mathcal{B}(G) &= \left( \frac{1}{|\mathcal{X}/\simeq_{1\text{WL}}(G)|} \sum_{G^* \in \mathcal{X}} \mathbb{1}_{G \simeq_{1\text{WL}} G^*} \right) \cdot \mathcal{B}(G) \\ &= \frac{1}{|\mathcal{X}/\simeq_{1\text{WL}}(G)|} \sum_{G^* \in \mathcal{X}} \mathcal{B}(G^*) \cdot \mathbb{1}_{G \simeq_{1\text{WL}} G^*} \\ &= \sum_{G^* \in \mathcal{X}} \frac{\mathcal{B}(G^*)}{|\mathcal{X}/\simeq_{1\text{WL}}(G^*)|} \cdot \varphi_{G^*}(G) \end{aligned} \tag{0.5}$$

where we denote with  $\mathcal{X}/\simeq_{1\text{WL}}(G^*)$  the set of all graphs  $G$  over  $\mathcal{X}$  that are equivalent to  $G^*$  according to the  $\simeq_{1\text{WL}}$  relation. Further, with Lemma 18 we know that for any  $G_1, G_2 \in \mathcal{X}$  with  $G_1 \simeq_{1\text{WL}} G_2 : \mathcal{B}(G_1) = \mathcal{B}(G_2)$ .

We can encode  $\mathcal{B}$  as stated in Equation 0.5 via a multilayer perceptron with  $\frac{\mathcal{B}(G^*)}{|\mathcal{X}/\simeq_{1\text{WL}}(G^*)|}$  being constants and  $\varphi_{G^*} \in \mathcal{C}$  encoding the indicator function. Combined with the Lemma 26, we can conclude that  $\mathcal{B}$  is computable by 1-WL+NN. Important to note, we can only do this since  $\mathcal{X}$  is finite, making the overall sum finite and the cardinality of  $\mathcal{X}/\simeq_{1\text{WL}}(G^*)$  well-defined for all graphs.  $\square$

**Part II.**

**Empirical Testing**

The empirical part is divided into blbnalanb

## 5. Testing Configuration

This section delves into the configuration and setup of our empirical testing. We begin by presenting our carefully selected datasets, which serve as the foundation for our evaluation. We highlight specific insights and characteristics of these datasets that make them compelling choices for our study. Afterward, we focus on the models we employ for testing and subsequent result comparison. We provide a comprehensive overview of the selected models, outlining their key features and motivations behind their inclusion in our evaluation. Subsequently, we provide a detailed description of the training pipeline and explain specific hyperparameters for which we will optimize these models.

### 5.1. Datasets

We will first explain our choice of datasets and introduce each dataset individually. Afterward, we will explore essential observations that are to be considered when assessing the actual empirical results.

#### Choice of Datasets

In selecting the datasets for our thesis, we adhered to two fundamental principles to ensure the robustness and diversity of our evaluation for GNN and 1-WL+NN models. The first principle focuses on using widely recognized benchmark datasets that have been extensively employed in previous studies. This principle enables us and readers to make meaningful comparisons with existing results. The second principle focuses on choosing datasets that are distinct from one another in terms of both their application domains and the way they encode information in graphs.

To fulfill the first principle, we opted for datasets from the TUDataset library. This library, curated by Morris et al. [2020], serves as a widely recognized standard for evaluating graph-related methods.

Regarding the second principle, we incorporated the insights from Liu et al. [2022], who developed a comprehensive taxonomy of common graph benchmark datasets. Their work examined the degree to which information is encoded in graph structures compared to node features with respect to solving the task of the datasets. Based on their taxonomy, they categorized datasets into three distinct classes:

1. Datasets in which the most crucial information for solving the task is contained in the node features.
2. Datasets similar to the first category, but with the significant exception that the node degree strongly correlates with the node features. In these datasets, utilizing simple structural information, such as computing the node degree, is as beneficial for solving the task as using the original node features.
3. Datasets where the most crucial information is encoded within the graph structure itself.

These categories help us understand how information is encoded in various datasets, such that we aim to choose datasets from all three categories.

As a result of considering these two principles, we selected the following datasets for our thesis: ENZYMES, IMDB-BINARY, MUTAG, NCI1, PROTEINS, and REDDIT-BINARY for classification tasks, and ALCHEMY and ZINC for regression tasks. For an overview of the elemental properties of each dataset, see Table 2. We will now shortly introduce each dataset individually:

ENZYMES, provided by Borgwardt et al. [2005], is a dataset consisting of proteins in their tertiary structure, categorized into six distinct enzyme classes. Each node represents a secondary structure, and has an edge to its three spatially closest nodes. Furthermore, each node feature encodes the type of secondary structure (*helix*, *sheet* or *turn*), as well as physical and chemical information.

IMDB-BINARY, provided by Yanardag and Vishwanathan [2015], is a dataset comprising ego networks. Each node in the network represents an actor/actress, and a unidirectional edge exists between two nodes if and only if the corresponding actors played together in a movie. The task involves determining whether each ego network’s genre is *action* or *romance*.

MUTAG, provided by Debnath et al. [1991], is a dataset comprising Nitroaromatic compounds. Each compound is represented by a graph in which nodes represent atoms, with their types encoded as node features, and edges represent atomic bonds. The task involves determining whether a given compound has a mutagenic effect on *Salmonella typhimurium* bacteria.

NCI1, provided by Wale et al. [2008], comprises graph representations of chemical compounds. In these graphs, nodes represent atoms, and edges represent atomic bonds. Moreover, the atom types are encoded in the node features. The overall task involves determining whether a given compound is active or inactive in inhibiting non-small cell lung cancer.

PROTEINS, provided by Borgwardt et al. [2005], contains proteins encoded similarly to ENZYMES. The task here is to determine whether each graph represents an enzyme.

REDDIT-BINARY, provided by Yanardag and Vishwanathan [2015], involves graphs that are derived from popular Reddit communities. The nodes in these graphs represent users who are active in the community, while the edges represent interactions between the users. The task is to identify whether a graph belongs to a community that is focused on questions and answers, or one that is focused on discussions.

ALCHEMY, provided by Chen et al. [2019], consists of organic molecules, with each node representing an atom and each edge representing an atomic bond. Additionally, each node feature encodes various properties for each atom, while each edge encodes the bond type and distance between atoms. The overall task is to compute 12 different continuous quantum mechanical properties for each graph.

ZINC, provided by Bresson and Laurent [2019] and Irwin et al. [2012], consists of molecular graphs where each node represents a heavy atom, and the corresponding node feature specifies its type. The edges in the graph encode the bonds between atoms, and their features further describe the type of bond. The task is to calculate a molecular property known as constrained solubility ( $\log P - \text{SA} - \text{cycle}$ ).

Table 2.: Dataset statistics and properties for graph-level prediction tasks. This table has been adapted from Morris et al. [2022].

Dataset	Properties						
	Number of graphs	Number of classes/targets	$\varnothing$ Number of nodes	$\varnothing$ Number of edges	Node labels	Edge labels	Taxonomy Category
Classification	ENZYMES	600	6	32.6	62.1	✓	✗
	IMDB-BINARY	1 000	2	19.8	96.5	✗	✗
	MUTAG	188	2	17.9	19.8	✓	✗
	NCI1	4 110	2	29.9	32.3	✓	✗
	PROTEINS	1 113	2	39.1	72.8	✓	✗
	REDDIT-BINARY	2 000	2	429.6	497.8	✗	✗
Reg.	ALCHEMY	202 579	12	10.1	10.4	✓	✓
	ZINC	249 456	1	23.1	24.9	✓	✓

### Analysis of the Datasets

To ensure the reliability and fairness of our evaluation, one of our initial steps was to assess the balance of our selected datasets for classification. We employed the normalized Shannon index to evaluate the balance of a dataset, where a value close to 0 indicates maximum imbalance, while a value close to 1 signifies perfect balance. See Appendix 1.1 in the Appendix for a formal definition of this metric.

Table 3.: An overview of the normalized Shannon index calculated for each dataset.

	Dataset					
	ENZYMES	IMDB-MULTI	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Shannon index	1	1	0.920	1	0.973	1

Table 3 provides an overview of the normalized Shannon index computed for each selected classification dataset. Upon analyzing the data, we observed that all the datasets exhibit high balance, as all their values are close to 1. This finding assures us that the datasets do not suffer significant class imbalances, which will remain important in the following section.

In addition to the balance of all datasets, another important aspect is understanding the upper bound of performance achievable by both GNN and 1-WL+NN models on these datasets. Unlike in other areas of machine learning where multilayer perceptron models can achieve near-perfect performance due to their universal approximation capabilities (Hornik [1991]), GNN and 1-WL+NN models have inherent limitations on their expressiveness. This restriction stems from the fact that the expressiveness of the 1-WL algorithm limits the performance of both frameworks. In detail, if the 1-WL algorithm can not distinguish a pair of graphs in a dataset, then neither a GNN nor a 1-WL+NN model can.

While we have pointed out that the 1-WL algorithm is, in general, quite powerful in distinguishing pairs of graphs in Section 2; we also explained that it is not complete. Therefore, assuming that GNN or 1-WL+NN models exist that can achieve almost perfect accuracy on all classification datasets is not reasonable. Thus, we calculated the theoretical maximum accuracy achievable by the perfect model for each dataset. In detail, we even investigated how

many iterations of the 1-WL algorithm it takes to achieve this accuracy. For a comprehensive overview of the accuracies achievable on all datasets, please refer to Table 4. In this table, we have included the accuracy achievable when running the 1-WL algorithm for 0 iterations, which essentially means taking the initial node features as the coloring for iteration 0 and assessing the expressiveness of this initial coloring.

Table 4.: An overview of the maximum theoretical classification accuracy achievable for each dataset based on the number of 1-WL iterations in percent. A hyphen “-” indicates that the maximum accuracy has converged with fewer iterations, implying that further iterations do not improve the accuracy.

1-WL	Dataset					
	ENZYMEs	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Iterations: 0	81.4	60.6	93.1	91.3	91.9	83.9
Iterations: 1	100.0	88.6	95.7	99.5	99.7	100.0
Iterations: 2	-	-	99.5	99.8	-	-
Iterations: 3	-	-	100.0	99.8	-	-
Iterations: 4	-	-	-	-	-	-
Max Accuracy	100.0	88.6	100.0	99.8	99.7	100.0

Upon examining the results, we observe that all datasets exhibit perfect or near-perfect theoretical classification accuracies. This result makes interpreting results obtained from actual 1-WL+NN or GNN models later on more straightforward. Additionally, the accuracy achievable after each number of iterations provides a theoretical lower limit on the number of message-passing layers a GNN must be composed of to be even capable of achieving this accuracy.

We have conducted the same analyzes on additional datasets, as their results might be valuable to the reader. For these, see Table B.1 in the Appendix.

## 5.2. Choice of Models

In selecting our models, we aimed to use techniques that are relatively generic and not highly specialized for any of the datasets, such that insights we gain upon analyzing the model’s performance can be generalized better. Consequently, we opted for a basic set of models to maintain simplicity and versatility.

### 1-WL+NN Models

Our primary consideration for the 1-WL+NN models revolves around choosing an encoding function, as all other components are fixed. To keep things straightforward, we decided to employ encoding functions consisting of two main components. Firstly, an optional preprocessing step that operates on the colors computed by the 1-WL algorithm. Secondly, a basic pooling function for transforming the color histogram into a fixed-size vector.

In more detail, the optional preprocessing step involves a simple look-up table. If utilized, this step maps each color injectively to a vector in the range of  $[-1, 1]^n$ , where the value of  $n$  is a hyperparameter. By encoding the color information into higher dimensions, this approach aims to enhance efficiency during subsequent processing steps. For the second step, the pooling

component, we selected elementary functions such as elementwise **Max**, **Mean**, and **Summation** (**Sum**). In summary, each of the **1-WL+NN** models can be uniquely identified by their encoding functions; therefore, we will refer to each model by their encoding function as follows:

$$\text{Embedding} - \{\text{Max}, \text{Mean}, \text{Sum}\} \quad \text{or} \quad \{\text{Max}, \text{Mean}, \text{Sum}\},$$

where “Embedding” indicates the use of the optional preprocessing step.

### GNN Models

As mentioned in the introduction to this section, our focus is on keeping the models basic. Hence, we opted for **GIN** by Xu et al. [2019], **GCN** by Kipf and Welling [2017], and **GAT** by Veličković et al. [2017] as the base architecture for the message-passing layers. Each of these architectures was chosen for specific reasons.

Firstly, we included **GIN** as an obvious candidate because it has been proven to be as expressive as the **1-WL** algorithm. This characteristic makes it interesting when comparing it to **1-WL+NN** models. Next, we also included **GCN** based on its good empirical success in recent years. Furthermore, the insights provided by Nikolentzos et al. [2023a] demonstrated that, in a certain sense, the node features computed by **GCN** and **GIN** are similar, making **GCN** a reasonable alternative to **GIN** in cases where the advantage of **GIN**’s expressiveness is not needed. Lastly, we added **GAT** as an alternative approach to the other two architectures. In detail, Nikolentzos et al. [2023a] also showed that the node features computed by **GAT** are entirely different from those computed by **GIN** or **GCN**.

Regarding the choice of Readout functions, we decided to utilize functions that are composed of two components. The first component is one of the elementwise pooling functions, such as **Max**, **Mean**, and **Sum**, to aggregate the information from a graph representation into a fixed-sized vector. Secondly, we incorporate a multilayer perceptron to process the aggregated information further, similarly to the **1-WL+NN** models.

In summary, each of the GNN models can be uniquely identified by their GNN architecture and the pooling function utilized; therefore, we will refer to each model by these two properties as follows:

$$\{\text{GAT}, \text{GCN}, \text{GIN}\} - \{\text{Max}, \text{Mean}, \text{Sum}\}.$$

Note that the selection of the Readout function creates similarities between the GNN and **1-WL+NN** models. These similarities play a crucial role in ensuring that any empirical differences observed between the two frameworks are not attributed to the utilization of more powerful techniques.

### 5.3. Experimental Setup

For the empirical testing, we took measures to ensure the reliability of our results in terms of hyperparameter configuration. Additionally, we aimed to ensure the reproducibility of the training pipeline for each model.

#### Testing Procedure

We started by conducting tests on the classification datasets, which serve as the basis for our evaluation. In our testing code, we employ a 10-fold cross-validation approach. Within this framework, we further partitioned the training data randomly, allocating 10 % of it for

validation purposes. Consequently, each training iteration consisted of 81 % of the original dataset as the training set, 9 % as the validation set, and 10 % as the test set. We repeat this testing procedure five times to mitigate the influence of randomness on the model’s performance. Ultimately, we recorded the mean accuracy on the test set along with its standard deviation as the primary performance metric of the model. Note that the use of standard cross-validation for generating the splits and the choice of mean accuracy as our primary metric is reasonable and justified as the datasets are highly balanced, as demonstrated in Section 2.

In the case of the regression datasets, we adopted a slightly different testing procedure. Due to their larger scale compared to the classification datasets, we opted for a more time-efficient approach. To achieve this, we utilized pre-existing training pipelines developed in Morris et al. [2020] and Morris et al. [2022]. These pipelines use a fixed training, validation, and test split, accelerating the testing process. Similar to the classification datasets, we performed five runs for each model configuration and conducted a hyperparameter sweep for all model configurations. We record the mean absolute error and its standard deviation, as well as the mean logarithmic absolute error and its standard deviation across all runs. Furthermore, we test two splits for each regression dataset: one utilizing only 10 000 samples for training, while the other split uses the entire training set.

## Hyperparameter Optimization

To ensure the comparability of our results with other works and limit the number of hyperparameters requiring optimization, we followed standard practices when configuring our models. Detailed information on all hyperparameters, including the ones we optimized, can be found in Table B.2 for the 1-WL+NN models used in classification tasks, Table B.3 for the GNN models employed in classification tasks, and Table B.4 for the 1-WL+NN models utilized in the regression task in the Appendix. We will shortly discuss the most critical parameters we optimized for:

Early in our investigations, we made a significant observation regarding the learning performance of 1-WL+NN models. We noticed considerable performance discrepancies when comparing 1-WL+NN models using the standard version of the 1-WL algorithm with the one from GNN’s. Consequently, we investigated the possibility of parametrizing the 1-WL algorithm to enhance control over the expressiveness of the colorings it computes. Specifically, we focused on two crucial parameters: 1) the number of iterations and 2) the usage of its convergence behavior as an early termination criterion.

The advantage of this more granular parametrization is that it enables us to apply the same number of 1-WL iterations to each graph processed by a 1-WL+NN model by deactivating the convergence behavior and fixing the number of iterations. By deactivating this behavior and keeping the number of iterations minimal, we observed that the number of colors used by the 1-WL algorithm remained contained. As a result, the MLP component of each 1-WL+NN model seems to generalize better, significantly improving performance. Therefore, one of the most crucial hyperparameters we optimized for all 1-WL+NN models was the number of 1-WL iterations. Additionally, if employed, the size of the dimension of the look-up table was another significant parameter to optimize.

In contrast, the hyperparameter selection for the GNN models was relatively less intricate. The key parameters of interest here are the number of message-passing layers and the size of the hidden dimensions used in each layer. The reason for this is that the number of layers directly corresponds to the expressiveness of the GNN, as demonstrated in the proof, while the size of the hidden dimensions enables easier learning by allowing the model to store more

information during graph processing.

### Implementation Details and Result Accessibility

The implementation of our models and training procedures was carried out using PYTHON 3.10 along with the open-source library PYTORCH<sup>6</sup> and its extension PYTORCH GEOMETRIC<sup>7</sup>. Moreover, we leveraged WEIGHTS&BIASES as our tool for coordinating and recording the results of the hyperparameter optimization. The code for our experiments is publicly available on GITHUB at <https://github.com/ericbill21/BachelorThesis>, and the corresponding results can be accessed via WEIGHTS&BIASES at <https://wandb.ai/eric-bill/BachelorThesisExperiments>. We conducted our tests on the RWTH High Performance Computing cluster by the RWTH Aachen University as well as on private hardware.

## 6. Empirical Testing

In this section, we present the empirical findings of our study. A total of 5 581 runs were conducted, with each run testing a specific model configuration on a designated dataset. These runs required a substantial amount of computation time, totaling 1 644 hours (over 70 days). To examine the classification accuracy achieved by the best-performing configuration of each model on each classification dataset, please refer to Table 5. For regression datasets, Table 6 provides an overview of the mean absolute error (MAE) for the best-performing configurations.

Due to the significantly larger size of the regression datasets compared to the classification datasets, running regression experiments required considerable time for each configuration. As a result, we had to limit the number of different configurations tested. For this reason, we employed existing training pipelines, as mentioned in the previous section, which allowed us to include the results of the **GINE- $\varepsilon$**  model proposed by Morris et al. [2020] and Morris et al. [2022] in Table 6. The **GINE- $\varepsilon$**  model utilizes the **GIN** architecture for message-passing layers, **Mean** as its pooling function, and a two-layer **MLP** for the final processing of its input graph. This design closely resembles our **GIN:Mean** model, with the crucial distinction being that **GINE- $\varepsilon$**  incorporates edge features in its computations.

The decision to exclude edge features from all our models, including all **1-WL+NN** models, was driven by two key factors. Firstly, our focus primarily revolved around the classification datasets, all of which do not encode any information in their edge features (refer to Table 2). Secondly, the regression datasets we utilized in our evaluation include continuous edge features, making the application of the **1-WL** algorithm more complex. Although some work exists on modifying the **1-WL** algorithm to incorporate continuous features, we opted for a more straightforward and more scalable approach by not considering edge features in the computations of all our models. Nonetheless, this limitation does not compromise the integrity of our results, as our main objective is to compare similar **GNN** and **1-WL+NN** models. The inclusion of **GINE- $\varepsilon$**  showcases the potential and highlights the significance of edge feature information for solving the dataset tasks of **ALCHEMY** and **ZINC**.

A notable finding when analyzing the presented results is that the performance of **1-WL+NN** models is comparable to that of **GNN** models. In fact, the best **1-WL+NN** models even

---

<sup>6</sup>A free and open-source machine learning framework that was originally developed by Meta AI and is now part of the Linux Foundation umbrella. <https://pytorch.org>

<sup>7</sup>An open-source library that extends PYTORCH and allows for easy development and training of graph neural networks. <https://pytorch-geometric.readthedocs.io>

outperform the GNN models in all classification datasets, with the exception of IMDB-BINARY and REDDIT-BINARY. These two datasets stand out among the classification datasets as they lack node features. We will investigate this insight later on in more detail. Moreover, our results align with the empirical findings of other GNN models in various studies. The accuracy reported here falls within a similar range as observed in works by Xu et al. [2019], Morris et al. [2022, 2020], Zhang et al. [2018].

In the subsequent subsection, we will further analyze these results. Specifically, we will investigate the tradeoff between generality and expressiveness in 1-WL+NN models, the learned aggregation functions of each model, the ability of GNN models to approximate 1-WL+NN models, the differences in their learning behaviors, and the impact of GNN models performance by preprocessing the data.

Table 5.: Overview of the mean classification accuracies achieved by the best configuration of each model for each dataset in percent and standard deviation.

Method	Dataset					
	ENZYME	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
1-WL+NN	Max	16.7 $\pm$ 4.2	52.0 $\pm$ 5.3	73.8 $\pm$ 12.4	58.6 $\pm$ 3.3	62.9 $\pm$ 4.9
	Mean	18.2 $\pm$ 4.8	59.4 $\pm$ 5.8	77.1 $\pm$ 11.5	64.0 $\pm$ 3.3	60.9 $\pm$ 4.5
	Sum	18.0 $\pm$ 6.2	57.5 $\pm$ 5.1	66.8 $\pm$ 13.9	56.9 $\pm$ 3.8	65.6 $\pm$ 4.8
	Embedding-Max	40.5 $\pm$ 7.4	69.4 $\pm$ 4.9	81.1 $\pm$ 11.2	82.7 $\pm$ 2.0	<b>75.2</b> $\pm$ 3.9
	Embedding-Mean	42.6 $\pm$ 9.0	<b>72.4</b> $\pm$ 4.1	84.1 $\pm$ 9.1	83.1 $\pm$ 1.9	72.3 $\pm$ 4.2
	Embedding-Sum	<b>48.3</b> $\pm$ 8.1	72.0 $\pm$ 3.8	<b>85.1</b> $\pm$ 8.6	<b>83.6</b> $\pm$ 2.2	75.2 $\pm$ 4.5
	GAT:Max	31.2 $\pm$ 6.0	70.7 $\pm$ 4.8	71.1 $\pm$ 12.2	58.0 $\pm$ 4.2	72.5 $\pm$ 5.1
	GAT:Mean	28.9 $\pm$ 5.9	70.9 $\pm$ 3.7	74.8 $\pm$ 9.1	66.1 $\pm$ 2.8	64.9 $\pm$ 6.4
	GAT:Sum	<b>34.4</b> $\pm$ 7.0	72.2 $\pm$ 4.5	82.1 $\pm$ 11.2	69.8 $\pm$ 2.6	73.4 $\pm$ 3.9
Graph Neural Networks	GCN:Max	33.1 $\pm$ 7.5	73.5 $\pm$ 4.1	74.5 $\pm$ 11.3	61.1 $\pm$ 3.6	69.8 $\pm$ 5.9
	GCN:Mean	29.9 $\pm$ 5.7	<b>74.7</b> $\pm$ 3.8	75.0 $\pm$ 10.4	68.9 $\pm$ 2.4	70.9 $\pm$ 5.2
	GCN:Sum	31.7 $\pm$ 7.2	73.0 $\pm$ 4.4	81.5 $\pm$ 10.3	70.4 $\pm$ 2.1	3.5 $\pm$ 3.9
	GIN:Max	29.2 $\pm$ 6.2	70.8 $\pm$ 4.7	77.3 $\pm$ 10.7	<b>79.9</b> $\pm$ 2.2	<b>74.3</b> $\pm$ 5.1
	GIN:Mean	31.7 $\pm$ 6.7	71.1 $\pm$ 5.4	82.4 $\pm$ 9.8	70.8 $\pm$ 2.2	72.0 $\pm$ 4.0
	GIN:Sum	28.9 $\pm$ 8.7	69.5 $\pm$ 4.8	<b>84.6</b> $\pm$ 8.7	70.8 $\pm$ 2.3	74.0 $\pm$ 4.3

Table 6.: Overview of the mean absolute error and the standard deviation (logMAE) on large-scale (multi-target) molecular regression tasks. The results marked by <sup>1</sup> are adopted from Morris et al. [2020] and by <sup>2</sup> from Morris et al. [2022].

Method	Dataset			
	ALCHEMY	ALCHEMY (10K)	ZINC	ZINC (10K)
GINE- $\varepsilon$ <sup>1,2</sup>	0.103 $\pm$ 0.001 -2.956 $\pm$ 0.029 <sup>1</sup>	0.180 $\pm$ 0.006 -1.958 $\pm$ 0.047 <sup>2</sup>	-	0.084 $\pm$ 0.004 <sup>1</sup>
GIN:Mean	0.643 $\pm$ 0.014 -0.505 $\pm$ 0.021	0.314 $\pm$ 0.004 -1.469 $\pm$ 0.044		0.341 $\pm$ 0.037
Embedding-Max	0.648 $\pm$ 0.003 -0.511 $\pm$ 0.009	0.409 $\pm$ 0.003 -1.023 $\pm$ 0.009	0.382 $\pm$ 0.005	0.659 $\pm$ 0.007
Embedding-Mean	0.617 $\pm$ 0.003 -0.564 $\pm$ 0.005	0.355 $\pm$ 0.004 -1.269 $\pm$ 0.020	<b>0.348</b> $\pm$ 0.010	0.484 $\pm$ 0.009
Embedding-Sum	<b>0.600</b> $\pm$ 0.004 -0.625 $\pm$ 0.032	<b>0.305</b> $\pm$ 0.001 -1.740 $\pm$ 0.042	0.384 $\pm$ 0.004	<b>0.465</b> $\pm$ 0.009

## 6.1. Fixed 1-WL Iterations: Tradeoff between Expressiveness and Generality

As discussed in Section 5.3, we explored constraining the expressiveness of the 1-WL algorithm by fixing the number of iterations it is applied to each graph in a 1-WL+NN model. The intention behind this was to ensure that the resulting colorings of graphs computed using a fixed 1-WL algorithm would fall within a similar color range, thereby limiting the total number of colors and avoiding outlier cases.

From a theoretical perspective, fixing the number of 1-WL iterations in a 1-WL+NN model only limits its expressiveness. For the remainder of this section, we will denote the number of 1-WL iterations by  $n_{\text{wl}} \in \mathbb{N}$ . For instance, let  $n_{\text{wl}} \in \mathbb{N}$  now be fixed. If the standard 1-WL algorithm converges on an input graph  $G$  after  $k' < k$  iterations, the coloring computed after  $n_{\text{wl}}$  iterations is as expressive as the one obtained from the standard 1-WL algorithm since the coloring convergence after  $k'$  iterations. On the contrary, if the standard 1-WL algorithm converges after  $k' > k$  iterations on an input graph  $G$ , the coloring obtained after  $n_{\text{wl}}$  iterations is less expressive and contains less information than the coloring derived from the standard 1-WL algorithm.

Comparing the performance of 1-WL+NN models utilizing the standard 1-WL algorithm to those using the parametrized version reveals a significant difference across all classification datasets, as seen in Table 7. This data confirms our belief that the standard 1-WL algorithm can be overly expressive for specific tasks, as evidenced by the considerable performance gap between the two types of the algorithm. Additionally, we included the mean number of iterations of the 1-WL algorithm applied on each dataset graph in the brackets in Table 4. This number remains a fixed constant for the models utilizing the parametrized version.

Property	Dataset					
	ENZYMEs	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Standard 1-WL	34.2 $\pm$ 6.7 (3.0)	67.0 $\pm$ 4.3 (1.2)	76.3 $\pm$ 9.4 (4.2)	(3.9)	71.4 $\pm$ 4.9 (3.0)	70.9 $\pm$ 3.8 (4.1)
Parametrized 1-WL	48.3 $\pm$ 8.1 (1)	72.4 $\pm$ 4.1 (1)	85.1 $\pm$ 8.6 (1)	83.6 $\pm$ 2.2 (3)	75.2 $\pm$ 3.9 (1)	78.4 $\pm$ 2.7 (1)

Table 7.: Comparison between the best performing 1-WL+NN models using the standard 1-WL algorithm and the one employing the parametrized version of it in percent and standard deviation. Additionally, we put the average number of iterations of the 1-WL algorithm in brackets for each model.

Building on these insights, investigating the optimal number of 1-WL iterations that lead to the best-performing models is another interesting observation. To illustrate its effect, we created Figure 7, which showcases the performance achieved by all models and configurations grouped by the number of 1-WL iterations for each dataset in a violin graph. Note that both classification and regression datasets are included in this figure. Depending on the dataset type, the y-axis represents either classification accuracy or test error, where higher accuracies imply better performance for classification tasks, and higher test errors imply poorer performance for regression tasks (and vice versa).

In general, the results indicate that the performance of 1-WL+NN models tends to improve as the number of 1-WL iterations decreases, except for NCI1, ALCHEMY and ZINC datasets.

Notably, for NCI1, the optimal number of iterations coincides with the number of 1-WL iterations needed for the maximum achievable accuracy (refer to Table 4). This correlation holds for all other classification datasets, except for MUTAG. However, the behavior observed in MUTAG can potentially be attributed to the fact that according to the introduced taxonomy

(Liu et al. [2022]) in Section 5.1, MUTAG can be effectively solved by simply replacing its node features with an encoding of the node degree. Since, from a theoretical perspective, the coloring computed after the first iteration of the 1-WL algorithm is usually an encoding of the node degree, this might explain the observed pattern.

Interpreting the results of the ZINC dataset is more complex and somewhat peculiar. While the smaller subset of the dataset, ZINC10K, clearly indicates that fewer 1-WL iterations yield better results, the overall ZINC dataset suggests the opposite. This discrepancy could be attributed to the limited number of runs conducted on the entire dataset, which were restricted due to scalability and time constraints. Therefore, these results should be interpreted with caution.

In conclusion, reducing the number of 1-WL iterations leads to improved generability and overall better performance of the models, with the clear exception of NCI1. However, as our results indicate, most 1-WL+NN models achieve optimal performance when employing only a single iteration of the 1-WL algorithm, which is essentially equivalent to encoding the node degree. This observation demonstrates that 1-WL+NN models do not require the full expressiveness of the 1-WL algorithm to achieve comparable performance to GNN models, which raises the question of whether GNN models, despite their theoretical ability to be highly expressive, also tend to rely primarily on node degree information for computation. We will explore this question further in Section TODO.

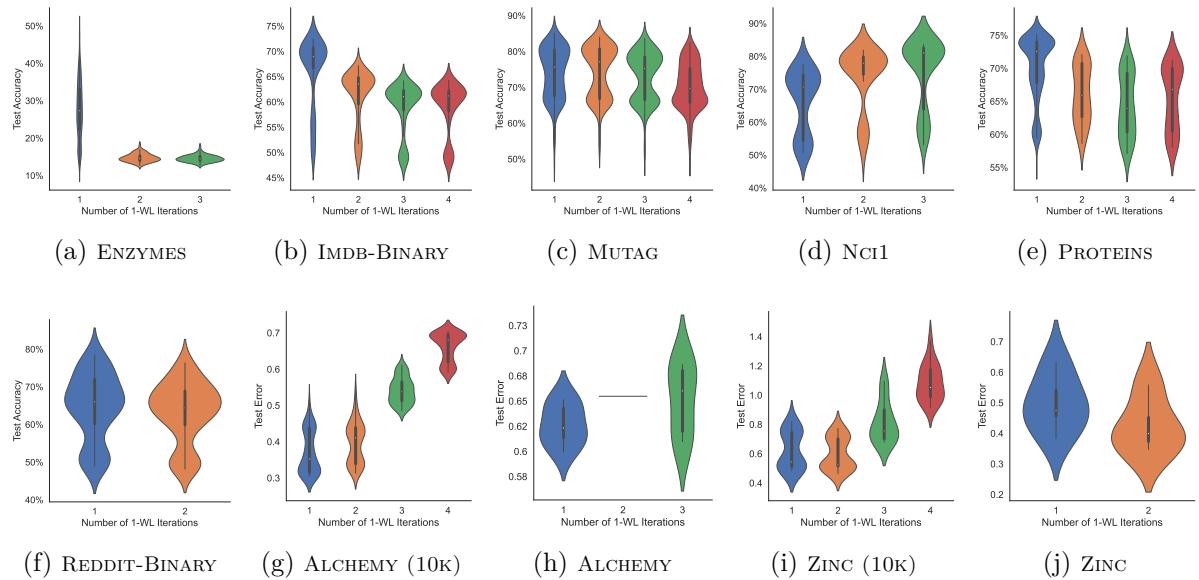


Figure 7.: Impact on performance by the number of iterations of the 1-WL algorithm.

## 6.2. Difference in Learning behavior

A problem with the 1-WL+NN models is that the 1-WL algorithm in particular is too expressive and this has a direct impact on the learning performance.

In order to grasp how expressive it is, we calculated for each dataset the ratio of unique colors compared to total amount of colors. For example, the dataset Mutag consists of 188 graphs in total. If we sum up the number of nodes of each graph, we end up with 3371 nodes in total. This is the upperbound of the total amount of colors the wl algorithm can apply use for

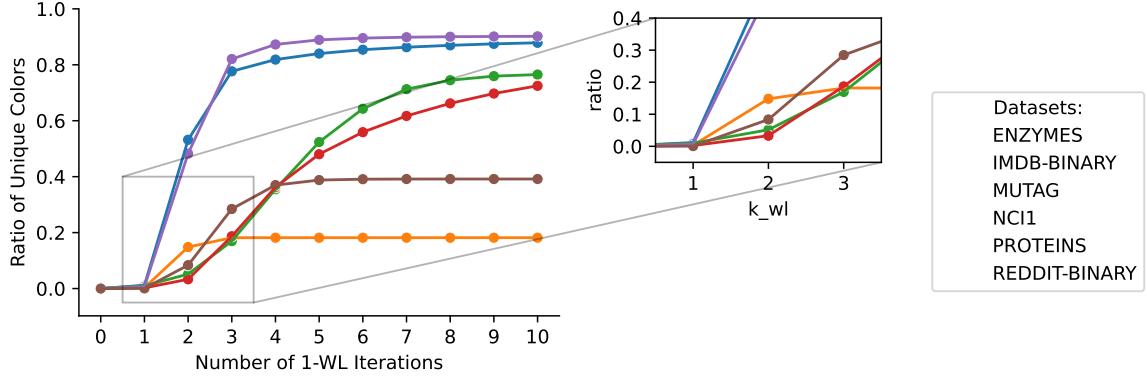


Figure 8.: Overview of the colors

calculating the colorings. If we now apply the wl alogirhm with a fixed number of wl iterations of 1 on Mutag, we end up with a coloring of each graph. If we now count the number of unique colors used we end up with 231 unique colors, which in comparison to the number of total nodes is very considerable, such that the ratio of used colors for the first iteration is 11.8%.

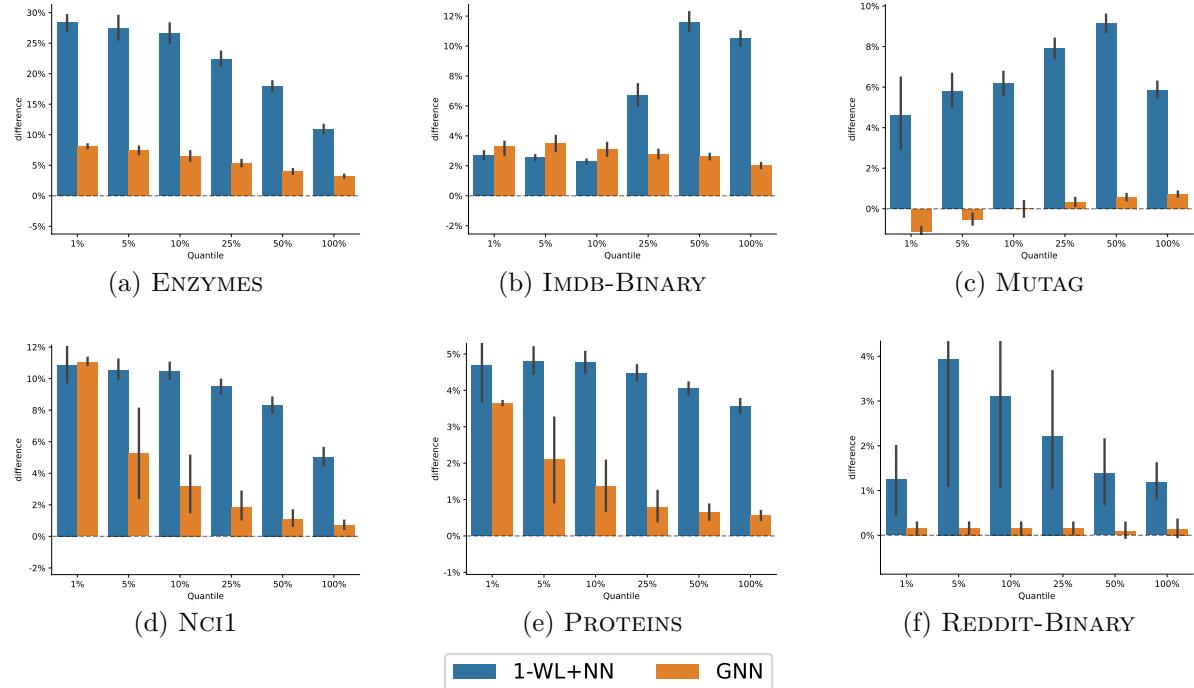


Figure 9.: Mean absolute difference of the classification accuracies of the training and testing set of each dataset. In detail, we compared the different quantiles of each performance.

**6.3. Aggregation Differnces**

**6.4. GNN Approximating 1-WL**

**6.5. Preprocessing Data for GNNs**

**7. Discussion**

**7.1. Learned Lessons**

**7.2. Future Work**

**8. Conclusion**

Method	Dataset					
	ENZYMES	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
1-WL+NN	MLP	48.3 $\pm$ 8.1	<b>72.4</b> $\pm$ 4.1	85.1 $\pm$ 8.6	83.6 $\pm$ 4.1	<b>75.2</b> $\pm$ 3.9
	SVM Linear	34.4 $\pm$ 5.5	71.2 $\pm$ 3.9	86.4 $\pm$ 8.9	83.4 $\pm$ 2.1	73.9 $\pm$ 4.1
	SVM RBF	45.0 $\pm$ 7.0	72.8 $\pm$ 4.3	83.2 $\pm$ 7.5	83.6 $\pm$ 1.9	75.2 $\pm$ 4.0
	k-NN	<b>56.3</b> $\pm$ 5.8 (k=1)	72.3 $\pm$ 4.1 (k=11)	<b>86.7</b> $\pm$ 7.7 (k=10)	<b>83.9</b> $\pm$ 1.8 (k=5)	73.9 $\pm$ 4.1 (k=19)
GNN	MLP	34.4 $\pm$ 7.0	<b>74.7</b> $\pm$ 3.8	84.6 $\pm$ 8.7	<b>79.9</b> $\pm$ 2.2	74.3 $\pm$ 5.1
	SVM Linear	33.2 $\pm$ 5.9	73.9 $\pm$ 4.2	51.9 $\pm$ 34.1	67.4 $\pm$ 2.2	74.7 $\pm$ 4.2
	SVM RBF	35.9 $\pm$ 6.0	74.1 $\pm$ 3.9	<b>86.0</b> $\pm$ 7.4	73.0 $\pm$ 1.9	74.6 $\pm$ 4.6
	k-NN	<b>51.6</b> $\pm$ 7.0 (k=1)	74.3 $\pm$ 4.0 (k=132)	88.3 $\pm$ 6.5 (k=38)	77.5 $\pm$ 1.7 (k=2)	<b>74.9</b> $\pm$ 4.3 (k=27)

Table 8.: Overview of the classification accuracies achieved by the best model for each dataset in percent and standard deviation. Additionally, the performance of each model was evaluated under alternative configurations, namely the substitution of the Multilayer Perceptron (MLP) with a Support Vector Machine employing either a linear kernel (SVM Linear) or the Radial Basis Function (SVM RBF), as well as the k-nearest neighbors algorithm (k-NN) with different values for  $k$ .

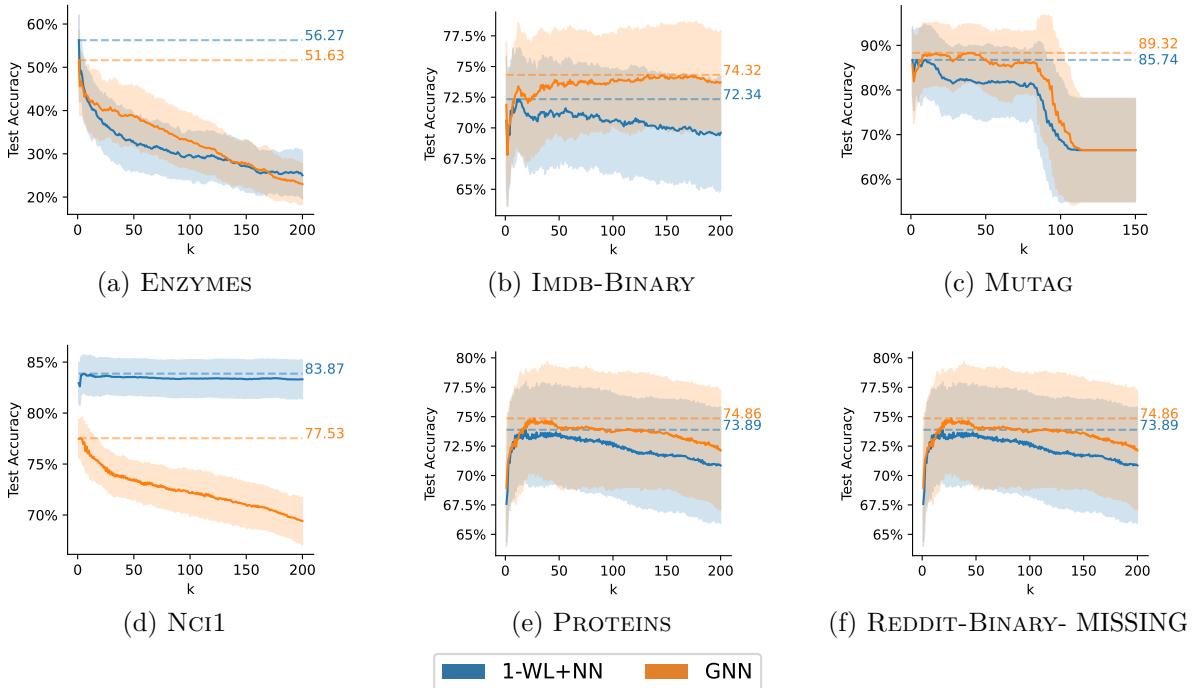


Figure 10.: Average classification accuracy achieved on each dataset by replacing the multilayer perceptron of the best-performing 1-WL+NN and GNN model with a classifier based on the  $k$ -nearest neighbors algorithm. We tested for different values of  $k$ .

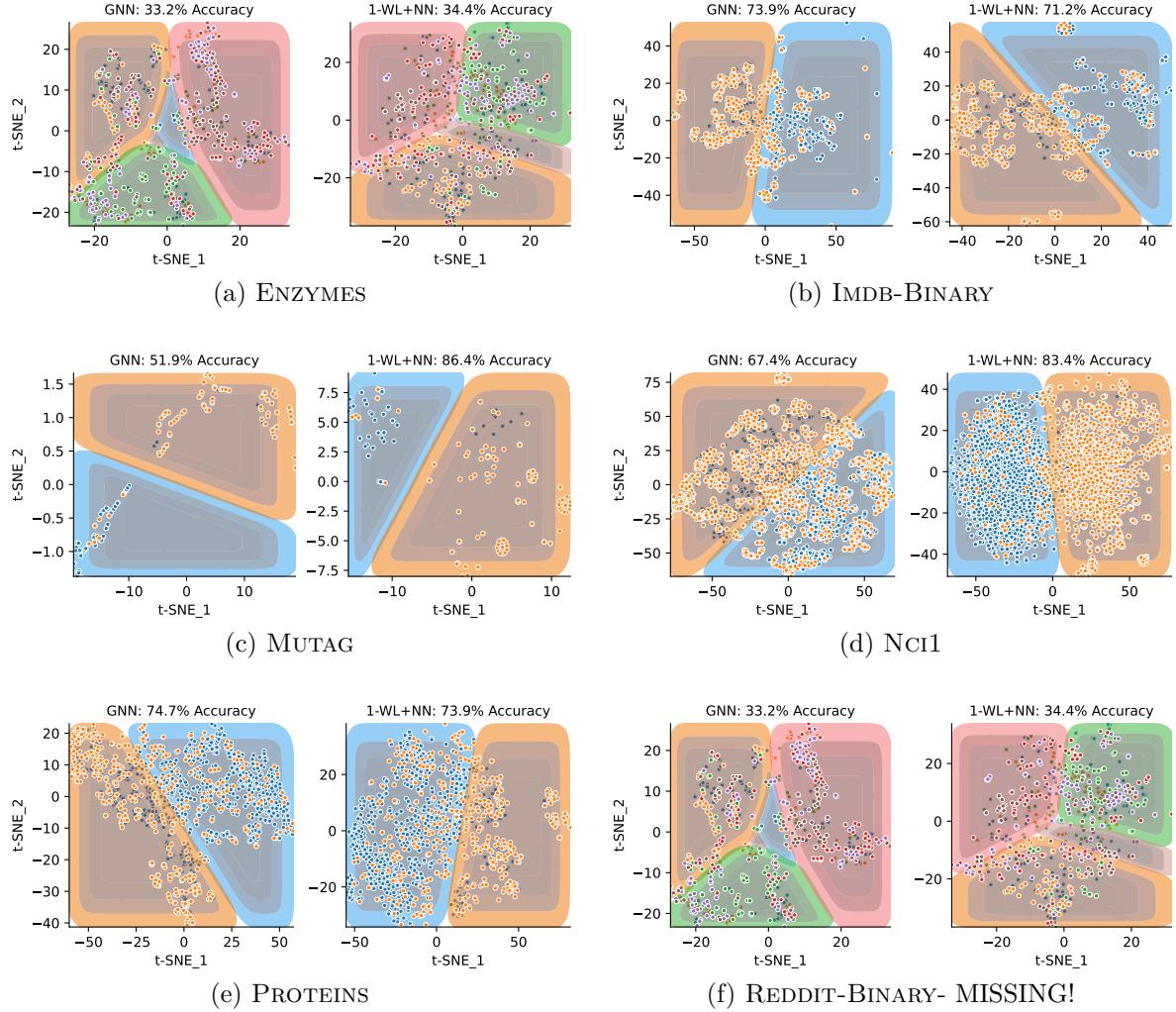


Figure 11.: Visualization of the decision boundary of each Support Vector Machine with a linear kernel using t-SNE.

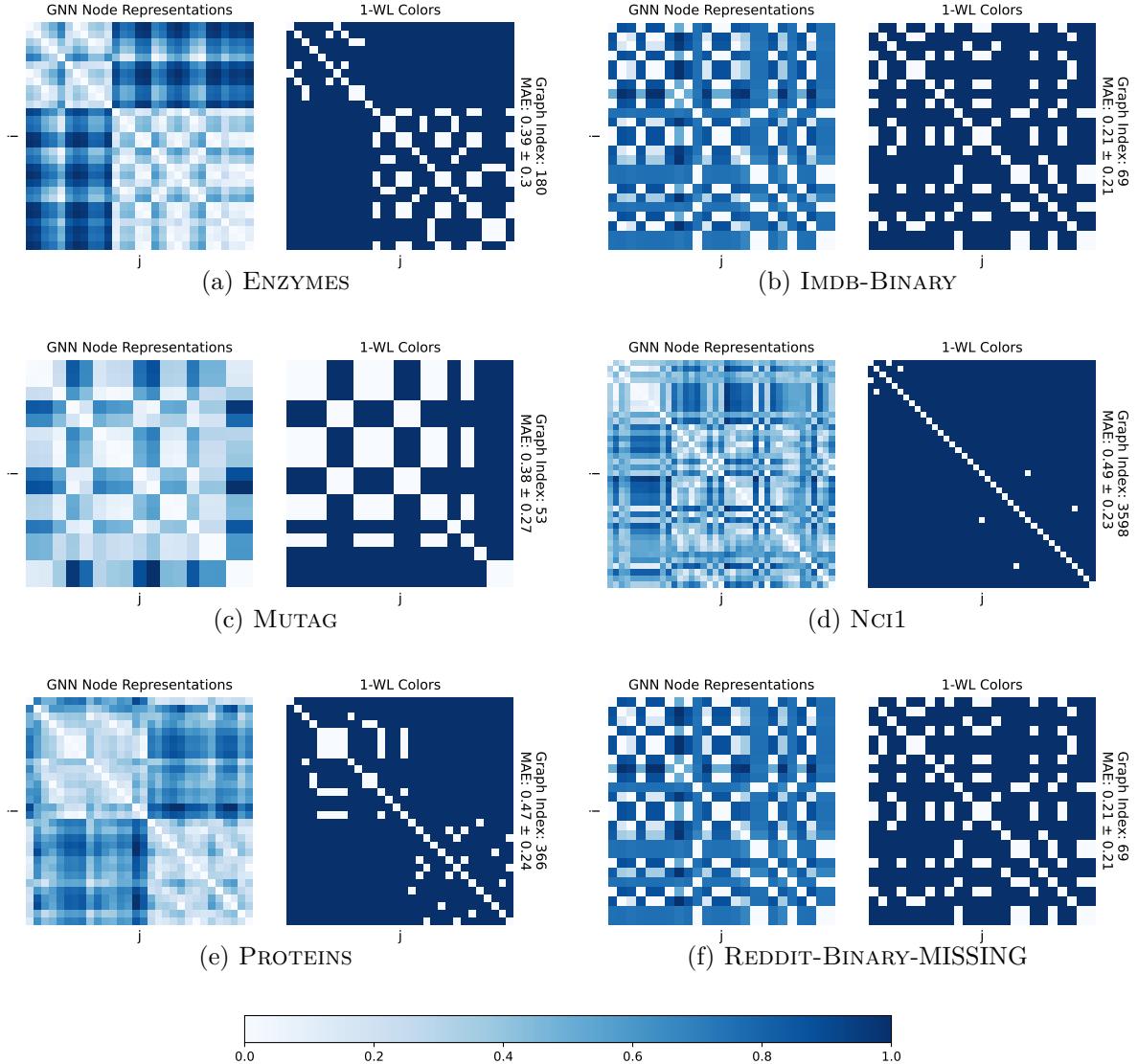


Figure 12.: Visualizing the performance of the best-performing GNN models of each dataset in approximating node colors computed by the 1-WL algorithm. Here we randomly sampled a single graph for each dataset and visualized the approximation performance. For each calculation of the 1-WL colors, we utilized the number of iterations of the 1-WL algorithm to be the same as the best-performing 1-WL+NN model of each dataset. Specifically, this was 1 for all datasets except for Nci1, where it was 3.

Property	Dataset					
	ENZYMES	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
MAE	0.49 $\pm$ 0.3	0.14 $\pm$ 0.15	0.42 $\pm$ 0.29	0.42 $\pm$ 0.22	0.49 $\pm$ 0.26	
Number of 1-WL colors	230	64	32	27252	296	

Table 9.: MAE of the approximation performance of the best-perfoming GNN in relation to the number of unique 1-WL colors for each dataset. In particular, in comparison to the colors computed by the 1-WL alogrithm with a single iteration.

Epsilon	Dataset					
	ENZYMES	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
$\epsilon = 0.001$	52.55	60.97	62.01			
$\epsilon = 0.01$	54.63	60.97	62.01			
$\epsilon = 0.05$	62.33	71.68	62.01			
$\epsilon = 0.1$	70.45	78.66	62.01			
Number of 1-WL colors	230	64	32	27252	296	

Table 10.: MAE of the approximation performance of the best-perfoming GNN in relation to the number of unique 1-WL colors for each dataset. In particular, in comparison to the colors computed by the 1-WL alogrithm with a single iteration.

# Bibliography

- [1] R. Abboud, I. I. Ceylan, M. Grohe, and T. Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.
- [2] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.
- [3] L. Babai. Lectures on graph isomorphism. University of Toronto, Department of Computer Science. Mimeographed lecture notes, October 1979, 1979.
- [4] L. Babai. Graph isomorphism in quasipolynomial time. In *ACM SIGACT Symposium on Theory of Computing*, pages 684–697, 2016.
- [5] L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *Symposium on Foundations of Computer Science*, pages 39–46, 1979.
- [6] D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Liò. Directional graph networks. In *International Conference on Machine Learning*, pages 748–758. PMLR, 2021.
- [7] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl\_1):i47–i56, 2005.
- [8] X. Bresson and T. Laurent. A two-step graph convolutional decoder for molecule generation. *arXiv preprint arXiv:1906.03412*, 2019.
- [9] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.
- [11] J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.
- [12] A. Cardon and M. Crochemore. Partitioning a graph in  $O(|A| \log_2 |V|)$ . *Theoretical Computer Science*, 19(1):85 – 98, 1982.
- [13] G. Chen, P. Chen, C.-Y. Hsieh, C.-K. Lee, B. Liao, R. Liao, W. Liu, J. Qiu, Q. Sun, J. Tang, et al. Alchemy: A quantum chemistry dataset for benchmarking ai models. *arXiv preprint arXiv:1906.09427*, 2019.

- [14] Z. Chen, S. Villar, L. Chen, and J. Bruna. On the equivalence between graph isomorphism testing and function approximation with GNNs. In *Advances in Neural Information Processing Systems*, pages 15868–15876, 2019.
- [15] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. 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, 1991.
- [16] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.
- [17] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.
- [18] F. Geerts. The expressive power of kth-order invariant graph networks, 2020.
- [19] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.
- [20] M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.
- [21] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.
- [22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [23] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [24] N. Immerman and E. Lander. *Describing Graphs: A First-Order Approach to Graph Canonization*, pages 59–81. Springer, 1990.
- [25] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.
- [26] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.
- [27] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [28] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

- [29] R. Liu, S. Cantürk, F. Wenkel, S. McGuire, X. Wang, A. Little, L. O’Bray, M. Perlmutter, B. Rieck, M. Hirn, et al. Taxonomy of benchmarks in graph representation learning. In *Learning on Graphs Conference*, pages 6–1. PMLR, 2022.
- [30] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.
- [31] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. Faster kernel for graphs with continuous attributes via hashing. In *IEEE International Conference on Data Mining*, pages 1095–1100, 2016.
- [32] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019.
- [33] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. *CoRR*, 2020.
- [34] C. Morris, G. Rattan, S. Kiefer, and S. Ravanbakhsh. Speqnets: Sparsity-aware permutation-equivariant graph networks. In *International Conference on Machine Learning*, pages 16017–16042. PMLR, 2022.
- [35] G. Nikolentzos, M. Chatzianastasis, and M. Vazirgiannis. What do gnns actually learn? towards understanding their representations. *arXiv preprint arXiv:2304.10851*, 2023a.
- [36] G. Nikolentzos, M. Chatzianastasis, and M. Vazirgiannis. Weisfeiler and leman go hyperbolic: Learning distance preserving node representations. In *International Conference on Artificial Intelligence and Statistics*, pages 1037–1054. PMLR, 2023b.
- [37] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
- [38] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [39] R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.
- [40] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.
- [41] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.
- [42] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.
- [43] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.
- [44] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

- [45] C. Vignac, A. Loukas, and P. Frossard. Building powerful and equivariant graph neural networks with structural message-passing. *Advances in neural information processing systems*, 33:14143–14155, 2020.
- [46] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14:347–375, 2008.
- [47] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.
- [48] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.
- [49] P. Yanardag and S. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.
- [50] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [51] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

# A. Appendix Part I

## 1. Figures and graphs

## 2. Proofs

### 2.1. Lemma 20: Composition Lemma for 1-WL+NN

Before we begin with the actual composition proof, we give a formal definition and notation for MLPs.

**Definition 28** (Multilayer Perceptron). Multilayer perceptrons are a class of functions from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ , with  $n, m \in \mathbb{N}$ . In this thesis, we define a multilayer perceptron as a finite sequence, such that a multilayer perceptron  $\text{MLP}$  is defined as  $\text{MLP} := (\text{MLP}_i)_{i \in [k]}$  where  $k$  is the number of layers. For every  $i \in [k]$ , the  $i$ .th layer of the  $\text{MLP}$  is the  $i$ .th item in the finite sequence  $\text{MLP}_i$ . Further, all layers are recursively defined as:

$$\begin{aligned}\text{MLP}_0(v) &:= v \\ \text{MLP}_{i+1}(v) &:= \sigma_i(W_i \cdot \text{MLP}_i(v) + b_i), \quad \forall i \in [k-1]\end{aligned}$$

where  $\sigma_i$  is an element wise activation function,  $W_i$  is the weight matrix and  $b_i$  the bias vector of layer  $i$ . Note, that for each  $W_i$ , the succeeding  $W_{i+1}$  must have the same number of columns as  $W_i$  has rows, in order to be well-defined. Similarly, for every layer  $i$ ,  $W_i$  and  $b_i$  have to have the same number of rows. Following this definition, when applying a  $\text{MLP}$  on input  $v \in \mathbb{R}^n$  it is  $\text{MLP}(v) := (\text{MLP})_k(v)$ .

*Proof of Lemma 20.* Let  $\mathcal{C}$  be a collection of functions computed by 1-WL+NN,  $h_1, \dots, h_n \in \mathcal{C}$ , and  $\text{MLP}^\bullet$  a multilayer perceptron. Further, let  $f_1, \dots, f_n$  be the encoding functions, as well as  $\text{MLP}_1, \dots, \text{MLP}_n$  be the multilayer perceptrons used by  $h_1, \dots, h_n$  respectively. As outlined above, we will now construct  $f^*$  and  $\text{MLP}^*$ , such that for all graphs  $G \in \mathcal{X}$ :

$$\text{MLP}^\bullet(h_1(G), \dots, h_n(G)) = \text{MLP}^* \circ f^*(\{\{1\text{-WL}(G)(v) \mid v \in V(G)\}\})$$

with which we can conclude that the composition of multiple functions computable by 1-WL+NN, is in fact also 1-WL+NN computable.

We define the new encoding function  $f^*$  to work as follows on an arbitrary input multiset  $M$ :

$$f^*(M) := \text{concat}\left(\begin{bmatrix} f_1(M) \\ \vdots \\ f_n(M) \end{bmatrix}\right),$$

where `concat` is the concatenation function, concatenating all encoding vectors to one single vector.

Adapt  
the  
notation  
for  
sequences  
 $(\text{MLP}_i)_{i \in [k]}$

Using the decomposition introduced in Definition 28, we can decompose each  $\text{MLP}_i$  for  $i \in [n]$  at layer  $j > 0$  as follows:  $(\text{MLP}_i)_j(v) := \sigma_{i,t}(W_j^i \cdot (\text{MLP}_i)_{j-1}(v) + b_j^i)$ . Using this notation we construct  $\text{MLP}^*$  as follows:

$$\begin{aligned} (\text{MLP}^*)_0(v) &:= v \\ (\text{MLP}^*)_{j+1}(v) &:= \sigma_j^*(W_j^* \cdot (\text{MLP}^*)_j(v) + \text{concat}\left(\begin{bmatrix} b_j^1 \\ \vdots \\ b_j^n \end{bmatrix}\right)) \quad , \forall j \in [k] \\ (\text{MLP}^*)_{j+k+1}(v) &:= (\text{MLP}^\bullet)_{j+1}(v) \quad , \forall j \in [k^\bullet - 1] \end{aligned}$$

where  $k$  is the maximum number of layers of the set of  $\text{MLP}_i$ 's, and  $k^\bullet$  is the number of layers of the given  $\text{MLP}^\bullet$ . Thereby, we define in the first equation line, that the start of the sequence is the input, with the second line, we construct the “simultaneous” execution of the  $\text{MLP}_i$ 's, and in the last equation line, we add the layers of the given  $\text{MLP}^\bullet$  to the end. Further, we define the weight matrix  $W_j^*$  as follows:

$$W_j^* := \begin{bmatrix} W_j^1 & 0 & \dots & 0 \\ 0 & W_j^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & W_j^n \end{bmatrix},$$

such that we build a new matrix where each individual weight matrix is placed along the diagonal. Here we denote with “0” zero matrices with the correct dimensions, such that  $W_j^*$  is well-defined. Important to note, should for an  $\text{MLP}_i$ ,  $W_j^i$  not exist, because it has less than  $j$  layers, we use for  $W_j^i$  the identity matrix  $I_m$  where  $m$  is the dimension of the output computed by  $\text{MLP}_i$ . And finally, we define the overall activation function  $\sigma_j^*$  as following:

$$\sigma_j^*(v) := \begin{bmatrix} \sigma_{1,j}(v[1]) \\ \vdots \\ \sigma_{1,j}(v[d_1]) \\ \vdots \\ \sigma_{n,j}(v[d_1 + \dots + d_{n-1} + 1]) \\ \vdots \\ \sigma_{n,j}(v[d_1 + \dots + d_n]) \end{bmatrix},$$

where  $d_i$  is the dimension of the output of  $\text{MLP}_i$  at layer  $j$ , and for the ease of readability we denote the  $i$ .th component of vector  $v$  here with  $v[i]$ . Thereby, we construct an activation function that applies each respective activation function of the  $\text{MLP}_i$ 's individually to their respective computation.  $\square$

## B. Appendix Part II

### 1. Definitions

#### 1.1. Definition of the normalized Shannon Index

We calculate the normalized Shannon index as follows:

$$-\frac{1}{\log_2(|C|)} \cdot \sum_{i \in C} \frac{n_i}{n} \cdot \log_2\left(\frac{n_i}{n}\right) \quad (\text{B.1})$$

where  $n$  is the total number of samples of the dataset,  $C$  is the set of all classes, and  $n_i$  is the number of samples of the class  $i \in C$ . As an example, for the dataset PROTEINS the variables are set to be the following:  $C = \{0, 1\}$  with  $n_0 = 663$  and  $n_1 = 450$ , so that  $n = 1113$ , yielding a rounded value of 0.973.

#### 1.2. Theoretical Maximum Accuracy Analysis

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Table B.1.: An overview of the maximum theoretical classification accuracy achievable for each dataset based on the number of 1-WL iterations in percent. A hyphen “-” indicates that the maximum accuracy has converged with fewer iterations, implying that further iterations do not improve the accuracy. “OOM” denotes out of memory error.

Datasets		Iterations of the 1-WL algorithm					
		0	1	2	3	4	5
Bioinformatics	DD	1.00	-	-	-	-	-
	ENZYMES	0.81	1.00	-	-	-	-
	KKI	1.00	-	-	-	-	-
	OHSU	1.00	-	-	-	-	-
	Peking_1	1.00	-	-	-	-	-
	PROTEINS	0.92	1.00	-	-	-	-
Small molecules	AIDS	1.00	1.00	-	-	-	-
	BZR	0.96	0.99	1.00	-	-	-
	COX2	0.93	0.96	0.99	1.00	-	-
	DHFR	0.92	0.95	1.00	1.00	-	-
	FRANKENSTEIN	0.63	0.77	0.88	0.89	0.89	-
	MUTAG	0.93	0.96	0.99	1.00	-	-
	NCI1	0.91	1.00	1.00	1.00	-	-
	NCI109	0.92	1.00	1.00	1.00	-	-
	PTC_MR	0.92	0.98	0.99	-	-	-
	COLLAB	0.61	0.98	-	-	-	-
Social networks	IMDB-BINARY	0.61	0.89	-	-	-	-
	IMDB-MULTI	0.44	0.63	-	-	-	-
	REDDIT-BINARY	0.84	1.00	-	-	-	-
	REDDIT-MULTI-5K	0.55	1.00	-	-	-	-
	REDDIT-MULTI-12K	0.36	OOM	OOM	OOM	OOM	OOM

### 1.3. Hyperparameter Configuration and Optimization

Table B.2.: 1-WL+NN!

Hyperparameter	Dataset					
	ENZYMES	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Batch Size	32	32	32	33	32	32
Learning Rate	$X \sim U(0.0001, 0.1)$					
Max Epochs	200	200	200	200	200	200
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Scheduler	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau
Number of 1-WL iterations	{1, 2, 3}	{1, 2, 3, 4}	{1, 2, 3, 4}	{1, 2, 3}	{1, 2, 3, 4}	{1, 2}
Use 1-WL-Convergence	False	False	False	False	False	False
MLP Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
MLP Normalization	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
MLP Number of Layers	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}
MLP Dropout	$X \sim U(0, 0.2)$					
Embedding Dimension	{None, 16, 32, 64, 128}					
Pooling function	{Max, Mean, Sum}					

Table B.3.: An Overview of hyperparameters we swooped and tested each dataset with.

Hyperparameter	Dataset					
	ENZYMEs	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Batch Size	32	32	32	{33, 129}	32	32
Learning Rate	$X \sim U(0.0001, 0.1)$					
Max Epochs	200	200	200	200	200	200
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Scheduler	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau
GNN Architecture	{GAT, GCN, GIN}					
GNN Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
GNN Dropout	$X \sim U(0, 0.2)$					
GNN Hidden Dimension	{16, 32, 64, 128}	{16, 32, 64, 128}	{16, 32, 64, 128}	{16, 32, 64, 128}	{16, 32, 64, 128}	{16, 32, 64, 128}
GNN Jumping-Knowledge	cat	cat	cat	cat	cat	cat
GNN Number of Layers	5	5	5	5	5	5
MLP Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
MLP Normalization	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
MLP Number of Layers	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}
MLP Dropout	$X \sim U(0, 0.2)$					
Pooling function	{Max, Mean, Sum}					

Table B.4.: 1-WL+NN!

Hyperparameter	Dataset			
	ALCHEMY	ALCHEMY(10K)	ZINC	ZINC(10K)
Batch Size	25	25	25	25
Learning Rate	0.001	0.001	0.001	0.001
Max Epochs	1000	1000	1000	1000
Optimizer	Adam	Adam	Adam	Adam
Scheduler	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau	ReduceLROnPlateau
Number of 1-WL iterations	{1, 2, 3}	{1, 2, 3, 4}	{1, 2, 3}	{1, 2, 3, 4}
Use 1-WL-Convergence	False	False	False	False
MLP Activation Function	ReLU	ReLU	ReLU	ReLU
MLP Normalization	BatchNorm	BatchNorm	BatchNorm	BatchNorm
MLP Number of Layers	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}	{2, 3, 4, 5}
MLP Dropout	$X \sim U(0, 0.2)$			
Embedding Dimension	{None, 16, 32, 64, 128}			
Pooling function	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}

## 2. Visualization of the Approximation Performance of GNNs

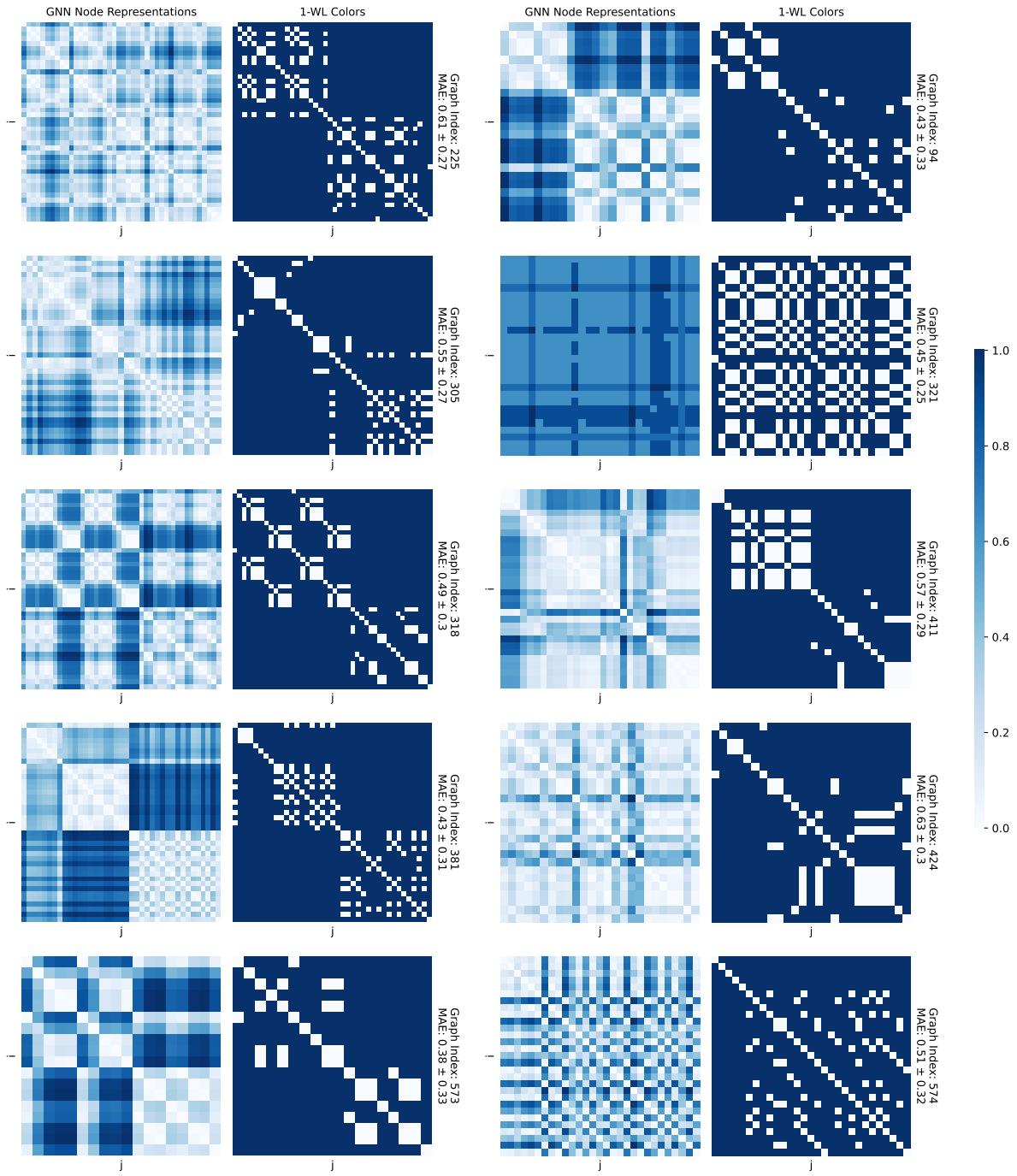


Figure B.1.: Visualizing the performance of the best performing GNN on the ENZYMES dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN’s test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is  $0.49 \pm 0.3$ .

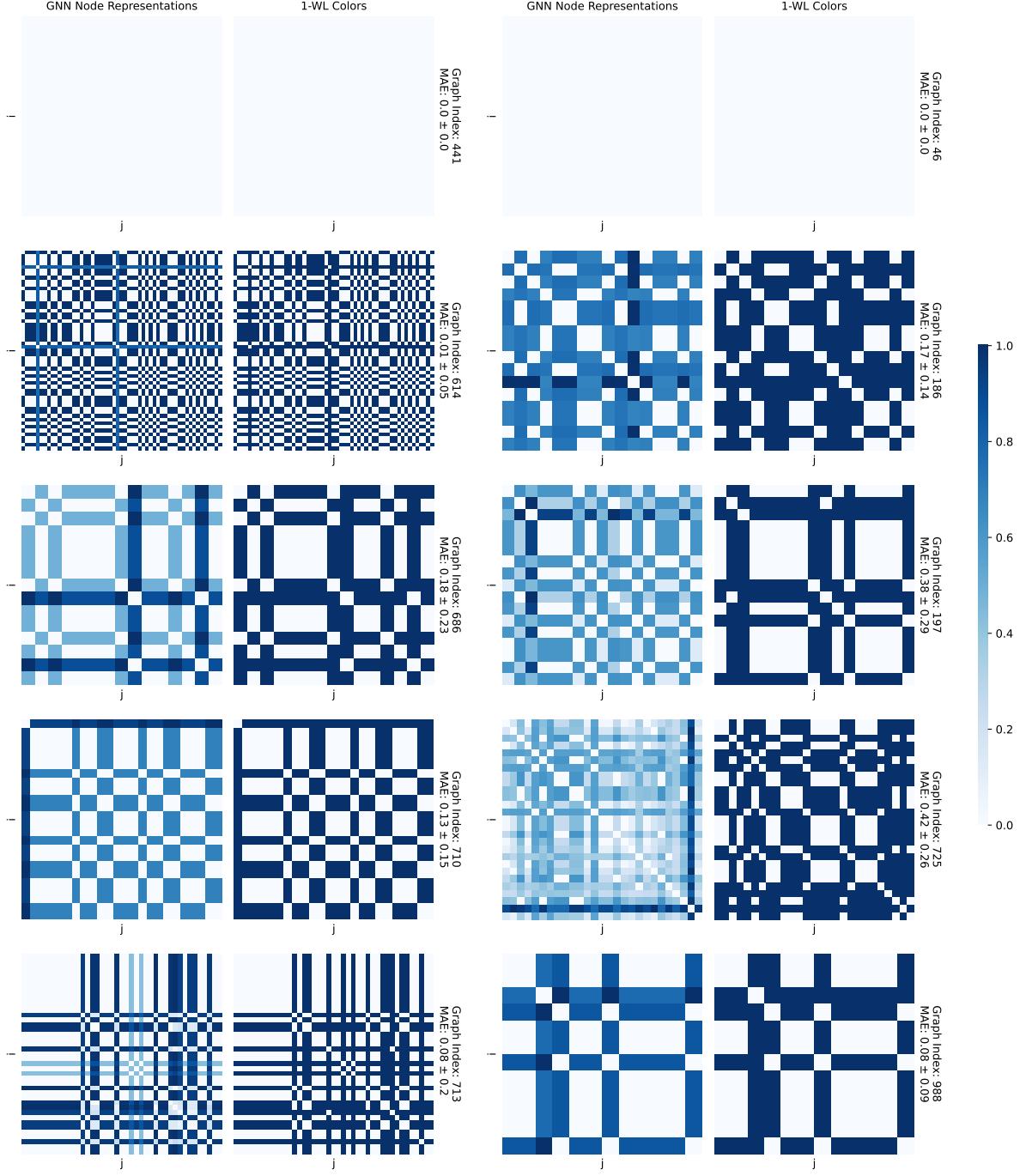


Figure B.2.: Visualizing the performance of the best performing GNN on the IMDB-BINARY dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN’s test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is  $0.14 \pm 0.15$ . Note that IMDB BINARY does not contain any node features, so we artificially initialize each node feature with a one-hot encoding of its degree.

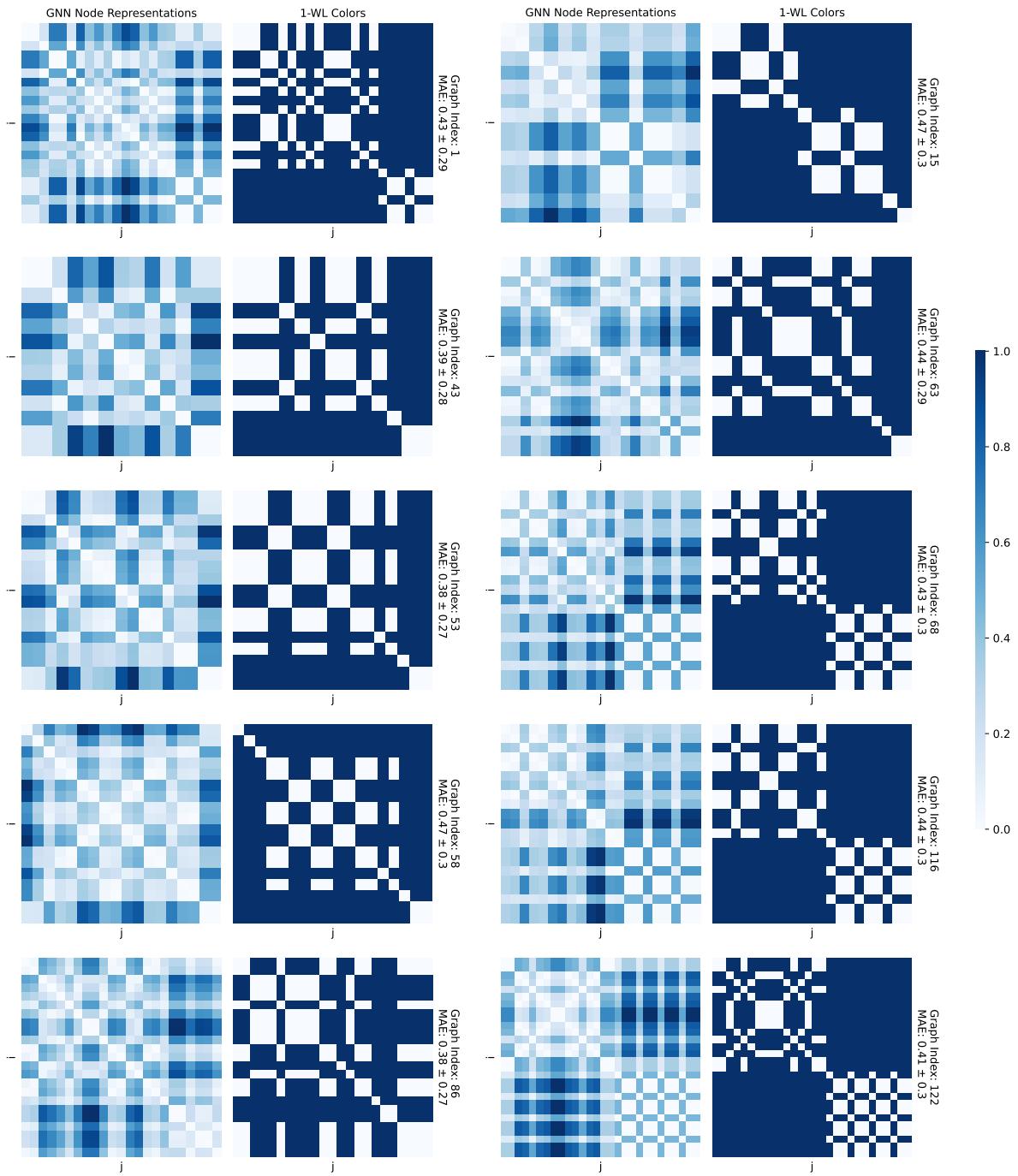


Figure B.3.: Visualizing the performance of the best performing GNN on the MUTAG dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN’s test set, and the 1-WL algorithm ran only for three iteration. The average error for the entire test set is  $0.42 \pm 0.29$ .

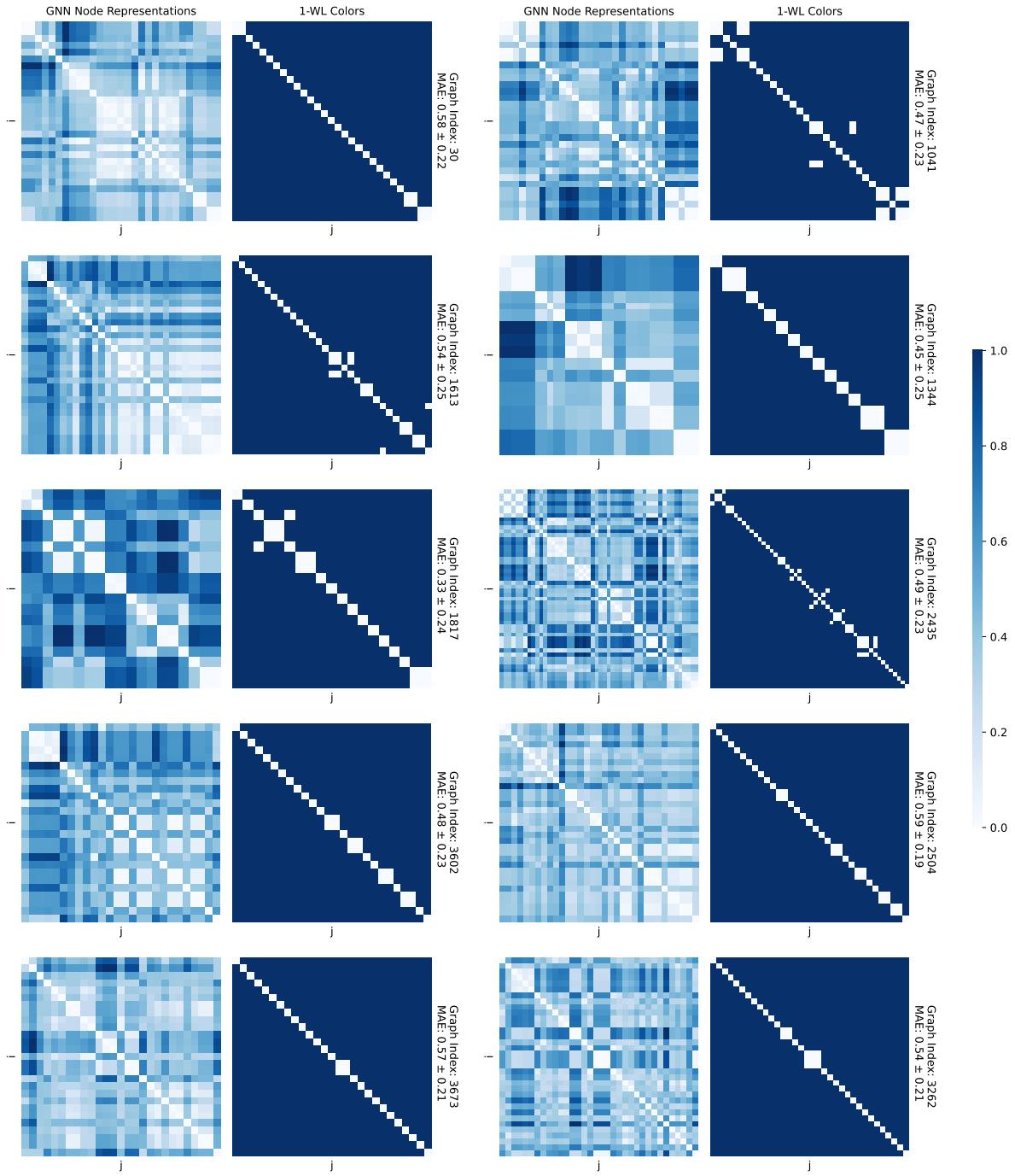


Figure B.4.: Visualizing the performance of the best performing GNN on the NCI1 dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN’s test set, and the 1-WL algorithm ran only for three iteration. The average error for the entire test set is  $0.50 \pm 0.24$ .

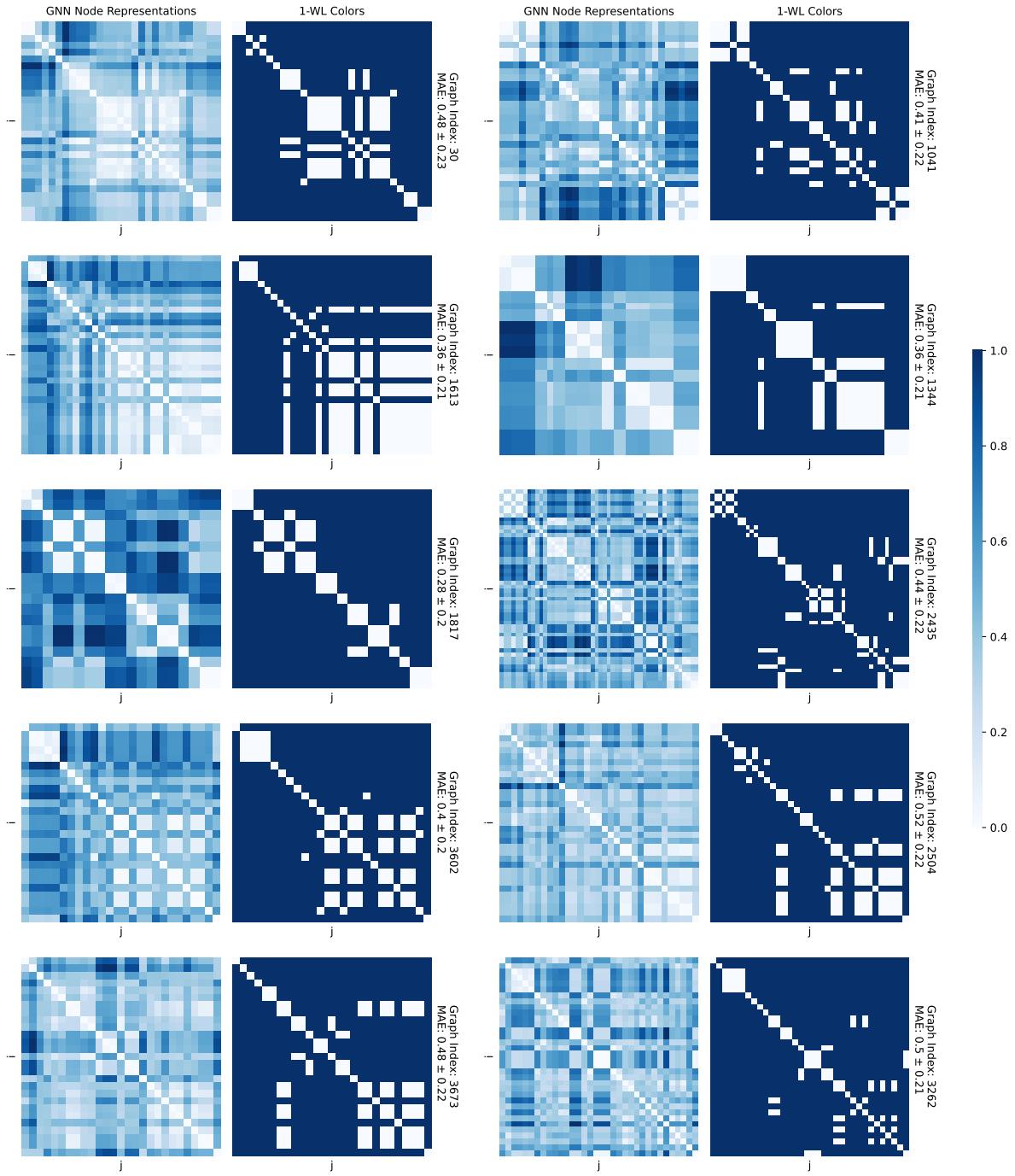


Figure B.5.: Visualizing the performance of the best performing GNN on the NCI1 dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN’s test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is  $0.42 \pm 0.22$ .

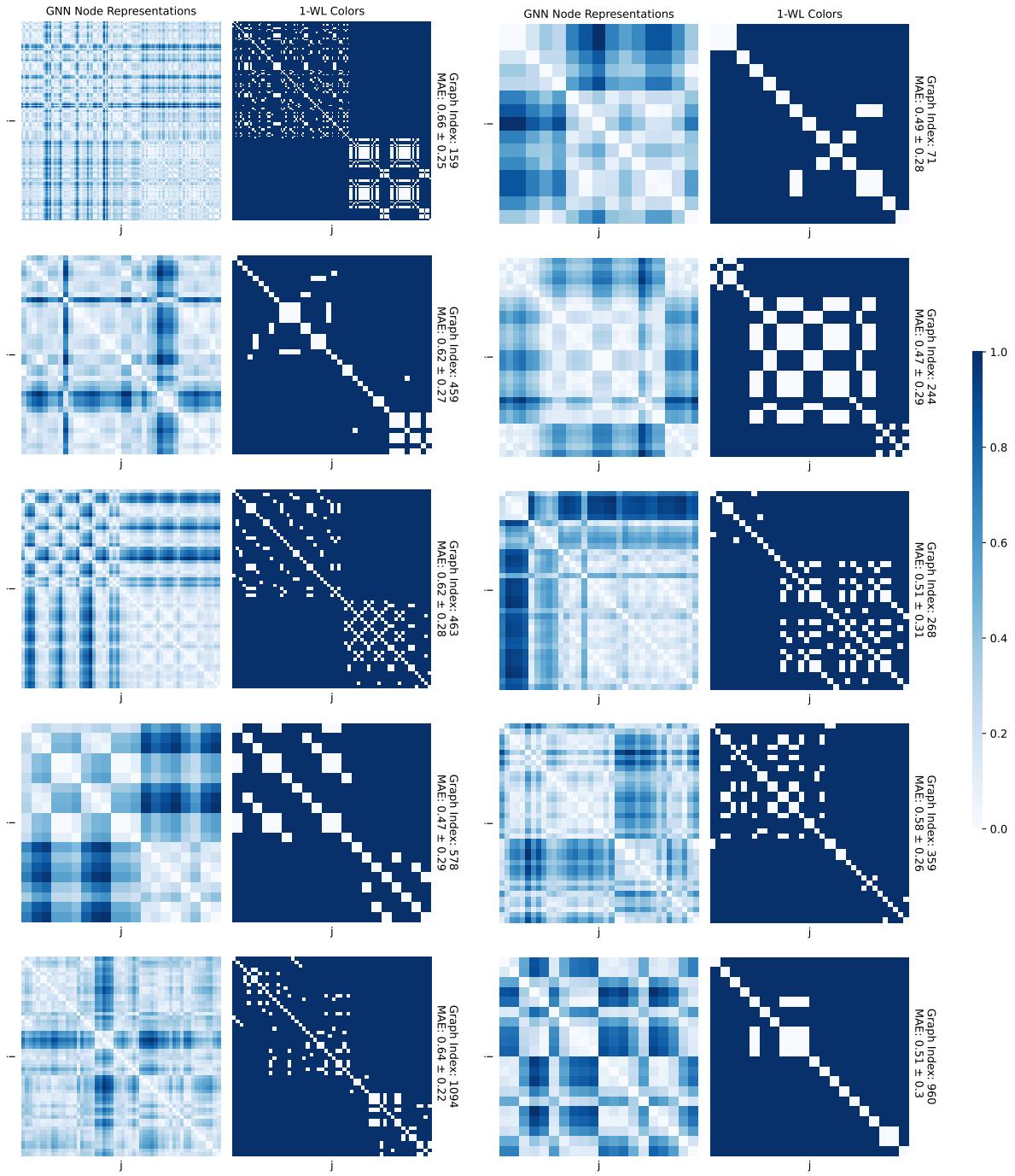


Figure B.6.: Visualizing the performance of the best performing GNN on the PROTEINS dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN’s test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is  $0.49 \pm 0.26$ .