Junior Professor for Machine Learning on Graphs

RWTH AACHEN UNIVERSITY

The present work was submitted to the Machine Learning on Graphs Group at the Chair of Computer Science 6, RWTH Aachen University

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

Supervisor:
Prof. Dr. rer. nat. Christopher Morris

Machine Learning on Graphs Group
RWTH Aachen University

July 26, 2023

Abstract

Lorem Ipsum :D

Acknowledgements

I want to express my gratitude to the people who helped me throughout this thesis. Their support and guidance made a huge difference, and I am genuinely thankful for their assistance.

First and foremost, I am extremely grateful for my esteemed supervisors, Christopher Morris and Luis Müller. Together, they formed a great team that not only introduced me to this captivating research field but also provided me with a research question that aligns perfectly with my passion. Their valuable insights, constant encouragement, and belief in my abilities have been the driving force behind this thesis.

In particular, I want to thank Christopher for his dedication and generosity in sharing his time and knowledge despite his commitments as a full-time professor. His responsiveness to my questions, provision of supplementary materials, and flexibility for meetings were invaluable throughout the entire thesis.

Additionally, I want to thank Luis Müller for being a sort of mentor for me. His guidance on best practices for empirical investigations in this research field, creative ideas for experiment setups along with his thorough proofreading and constructive feedback, have significantly enhanced the academic rigor of this work.

The most fruitful moments of this journey were undoubtedly the countless meetings with both Christopher and Luis. Through these discussions, we explored the progress of my thesis, addressed theoretical limitations in the proofs, and delved into intermediate empirical results and their interpretations. Each interaction left me with renewed motivation, making the overall pursuit of this thesis a challenging yet profoundly enjoyable experience.

I also want to thank the RWTH Aachen High Performance Computing cluster for providing the necessary computational resources for my experiments. Without these resources, the research goals of my thesis would not have been possible.

Lastly, I want to thank my parents and family for their unwavering support throughout my studies. Their encouragement means a lot to me and has been a great source of motivation.

Contents

1. Introduction

This work aims to shed light on the representations learned by Graph Neural Networks. 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

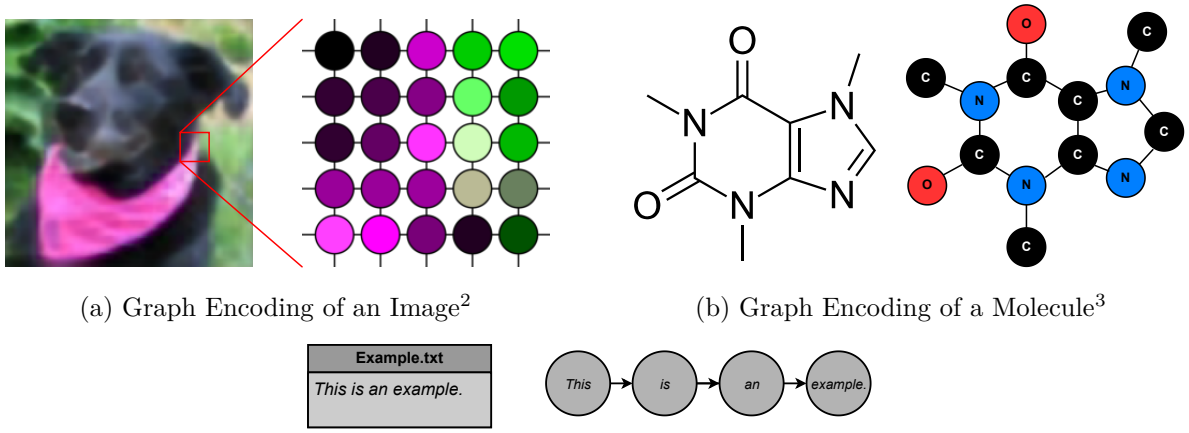
Graphs are ubiquitous in various fields of life. Despite not always being explicitly identified as such, the graph data model's flexibility and simplicity make it an effective tool for modeling a diverse range of data. Examples of graph modeling applications include unexpected instances, such as modeling 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 Weisfeiler-Leman[1] 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 that both methods are, to some extent, equivalent in their capacity to capture information in a graph.

[1] 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) Graph Encoding of an Image[2]

(b) Graph Encoding of a Molecule[3]

(c) Graph Encoding of a Text File.

Figure 1.: An overview of three examples of how graphs can be used to encode information across different domains. For each example, the conventional domain-specific encodings are visualized on the left, while on the right, we showcase how a graph can encode the same information. Note that these examples are just a sample; in actual practice, more detailed encodings are usually utilized to capture additional information.[4]

Despite the remarkable empirical performance of GNNs, particularly compared to conventional graph kernel functions Morris et al. [2020], and the theoretical understanding of their ability to distinguish non-isomorphic graphs, there remains a limited understanding of the learned representations that drive this empirical success. This work aims to delve into these representations, seeking deeper insights into the factors contributing to the efficacy of GNNs.

Footnotes are wrong!

1.2. Methology

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 that both frameworks, 1-WL+NN and GNN, are theoretically equivalent, such that each function computed by a 1-WL+NN model 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 class labels of the task, such that the model makes good predictions. In contrast, a GNN first must learn how to process the information in a

[2]The image of a dog is from the CIFAR-10 collection made available by Krizhevsky et al. [2009].
[3]The illustration of the skeletal formula of caffeine is taken from https://commons.wikimedia.org.
[4]All graphics were created using the free open source platform https://www.draw.io.

graph effectively and then find common patterns in its computed representation that correlate with the class labels so that the model can make good predictions.

To put both learning behaviors into perspective on a high level, while a 1-WL+NN model is given a maximally informative representation and needs to find the essential information in this representation to make a good prediction, a GNN works the other way around since it first has to learn how to effectively compute good representations of a graph and then leverage this information for good predictions. Despite their difference in learning behaviors, both methods can compute the same functions, making a fascinating comparison of their empirical performance.

Therefore, 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.

1.3. Research Questions & Contributions

1. 1-WL+NN as a framework for analyzing GNNs. We will show the theoretical equivalence and show that both methods share many emprical similarities, such that this tool is a great for analyzing various aspects of a GNN.
2. We will show

1.4. Outline

For 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 presents our different experiments and their empirical results, in which we use the 1-WL+NN framework as a tool to analyze the representation learned by a GNN.

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. Preceding, in Section 4, we will introduce two 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 is dedicated to our empirical experiments. We begin with Section 5, explaining the experimental setup and our experiment choices. In detail, we will discuss the choice of benchmarking datasets, GNN models, and 1-WL+NN models, along with an explanation of our general testing procedure. In Section 6, we present the results of our experiments and delve into further analyses of certain aspects of GNN and 1-WL+NN models. In particular, we will investigate the representations computed by GNNs and try to infer common patterns that occurred across multiple datasets. Finally, the thesis concludes with a final discussion in Section 7, where we summarize our findings, address the limitations of this work, and offer recommendations for future research.

2. Background and Related Work

In this section, we will briefly introduce the foundation of our research by explaining the origins of the two frameworks, 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 much 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.

This algorithm functions by assigning the same color to all nodes that meet two criteria: 1) they already share the same color, and 2) each color appears equally often in the set of the node's direct neighbors. The algorithm continues until the number of colors changes in each iteration. For determining whether two graphs are non-isomorphic, the heuristic is applied to both graphs simultaneously. The heuristic 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 Section 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][5]. 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 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, researchers faced challenges in effectively adapting these techniques to graphs of diverse sizes and complexities in the early stages. 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.

[5] 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.

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 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 GNN models, 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. The computation of each new node feature involves aggregating all the features of the neighboring nodes and the node's own feature. After applying each layer of a GNN model, a representation of the entire graph is obtained by applying a pooling function (e.g. Ying et al. [2018]). 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 in Section 3.5; 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 GNNs 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] has shown that this architecture imposes a hierarchy over $k \in \mathbb{N}$ that 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 authors, 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 GNN models and not be generalized. In another publication, Nikolentzos et al. [2023a] presented mathematical proof and empirical confirmation showing how much structural information is encoded by modern GNN models. Their 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 layers of message-passing with features that relate with the number of walks of length k over the input graph form each node, disregarding the local structure within the nodes are contained.

Part I.

Theoretical Equivalence

This part of the thesis focuses on the equivalence between 1-WL+NN and GNN. We will begin by providing a preliminary section that formalizes all the concepts used in the proof and introduces a general notation. Afterward, we will dedicate a separate section to present and prove three theorems. These theorems combined conclude the equivalence.

3. Preliminaries

This section will introduce and formalizes all concepts used throughout the proof and the rest of the thesis. 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 $\mathbb{N} := \{0, 1, 2, \ldots\}$. By $[n]$, we denote the set $\{0, \ldots, n\} \subset \mathbb{N}$ for each $n \in \mathbb{N}$. Further, with $\{\!\{\ldots\}\!\}$, we denote a multiset formally defined as a 2-tuple (X, m), where X is a set of all unique elements and $m : X \to \mathbb{N}_{\geq 1}$ 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. Starting with the definition of a graph.

Definition 1 (Graph)**.** A graph G is defined as a 3-tuple denoted by $G \coloneqq (V, E, l)$. 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 $l : M \to \Sigma$. The domain M of the labeling function can be either V, $V \cup E$, or E, and the codomain Σ is an alphabet with $\Sigma \subseteq \mathbb{N}^k$, where $k \in \mathbb{N}$ is arbitrary. 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 Σ. In detail, if $k = 1$, we usually refer to the values as labels, otherwise as features. 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 $E \subseteq \{(v, u) \mid v, u \in V, v \neq u\}$ 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 $V(G)$ 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. Further, with $\mathcal{N}(v)$ for $v \in V(G)$ we denote the set of neighbors of v defined as $\mathcal{N}(v) \coloneqq \{u \mid (u, v) \in E(G)\}$, and with $d(v)$ for $v \in V(G)$ the degree of node v, defined as $d(v) := |\mathcal{N}(v)|$.

We continue with the definition of a graph coloring.

Definition 2 (Graph Coloring)**.** A coloring of a Graph G is a function $C : V(G) \to \mathbb{N}$ 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 $\text{hist}_{G,C}$ 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, $\forall c \in \mathbb{N} : \text{hist}_{G,C}(c) \coloneqq |\{v \in V(G) \mid C(v) = c\}| = |C^{-1}(c)|$.

Permutation-invariance and -equivariance

We use S_n to denote the symmetric group over the elements $[n]$ for any $n \in \mathbb{N}$. 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 $G_\pi \coloneqq \pi \cdot G$ where $V(G_\pi) = \{\pi(0), \ldots, \pi(n)\}$ 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.

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

Definition 4 (Permuation Equivariant)**.** Let $f : \mathcal{G} \to \mathcal{Y}$ be an arbitrary function, then f is *permuation-equivariant* if and only if for all $G \in \mathcal{G}$, where $n_G \coloneqq |V(G)|$ 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: the coloring algorithm and the graph isomorphism test. We will introduce each part individually and present some implications afterward.

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 procedure and provide an illustrative 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 $C_i : V(G) \to \mathbb{N}$. In the initial iteration $i = 0$, the coloring is set to $C_0 = l$ if l exists. Otherwise, for all $v \in V(G) : C_0(v) = c$ with $c \in \mathbb{N}$ being an arbitrary but fixed constant. 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 $C_\infty \coloneqq C_i$ as the stable coloring, such that $\text{1-WL}(G) \coloneqq C_\infty$.

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_∞ can be computed in time $\mathcal{O}(|V(G)| + |E(G)| \cdot \log |V(G)|)$.

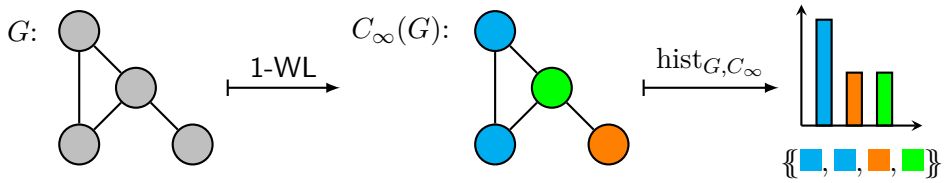


Figure 2.: An example of the final coloring computed by applying the 1-WL algorithm on the graph G. The graph G consists of 4 nodes with all their labels being set to the same color.

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 Part II.

The Weisfeiler-Leman Graph Isomorphism Test

The isomorphism test uses the 1-WL coloring algorithm and is defined as follows.

Definition 6 (1-WL Isomorphism Test)**.** To determine if two graphs $G, H \in \mathcal{G}$ are non-isomorphic ($G \ncong H$), 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:

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

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 for a straightforward example of where this test fails that was discovered and proven by Cai et al. [1992].

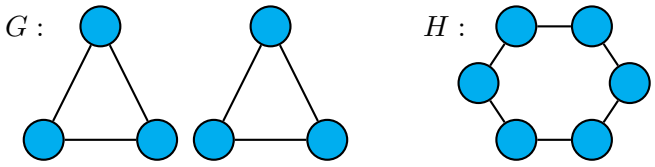


Figure 3.: This is an example of two graphs G and H that are non-isomorphic but cannot be distinguished by the 1-WL isomorphism test.

Implications of the 1-WL Algorithm

One implication of the 1-WL algorithm and its isomorphism test is that, due to it not being complete for solving the *graph isomorphism problem*, it gives rise to a related but weaker relation than the isomorphism relation ($\simeq$). We define this relation as follows.

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

The $\simeq_{1\text{WL}}$ 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 $\mathcal{X}/\!\simeq_{1\text{WL}}(G) := \{G' \in \mathcal{X} \mid G \simeq_{1\text{WL}} G'\}$ its equivalence class.

Similarly, we define the notion 1-WL-Discriminating for collections of permutation invariant functions.

Definition 8 (1-WL-Discriminating)**.** Let $\mathcal{X} \subseteq \mathcal{G}$. Further, 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 ($G_1 \not\simeq_{1\text{WL}} G_2$), there exists a function $h_{G_1,G_2} \in \mathcal{C}$ such that $h_{G_1,G_2}(G_1) \neq h_{G_1,G_2}(G_2)$.

3.4. 1-WL+NN

As the Section 2.1 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 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_{enc} operating on graph colorings, and 3. an arbitrary multilayer perceptron MLP. In detail, a 1-WL+NN model computes the function $\mathcal{B}$, that is defined as follows:

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

where 1-WL(G) is the coloring computed by the 1-WL algorithm when applied on G, and $k \in \mathbb{N}_{\geq 1}$ is a freely selectable hyperparameter. 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-level tasks by applying the encoding function f_{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-level 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 be straightforwardly adapted to 1-WL+NN models designed for node- or edge-level tasks.

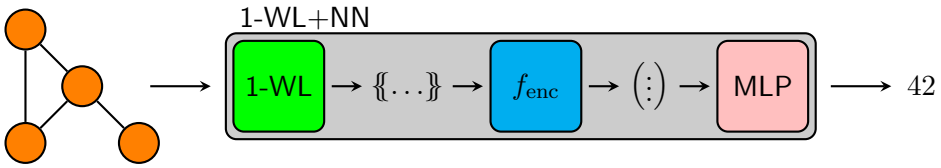


Figure 4.: This simplified illustration explains the components that make up a 1-WL+NN model and how each one processes the input further. In detail, the model takes the graph on the left as input and first applies the 1-WL algorithm, thereby obtaining a multiset of the colors assigned by the algorithm. Then the encoding function f_{enc} is applied, resulting in a fixed-sized vector that is 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 with 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 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 function of the format $f^{(0)} : V(G) \to \mathbb{R}^{1\times d}$ 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:

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

where $f^{(t)}_{\text{merge}}$ is an arbitrary function that maps the aforementioned tuple to a vector, effectively "merging" them, while $f^{(t)}_{\text{agg}}$ 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-sized output vector for the entire graph.

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

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

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 $f_{\text{merge}}^{(t)}$ and $f_{\text{agg}}^{(t)}$ functions for all $t \in [T]$, along with the final Readout function, to be differentiable.

Note that, due to our definition of the "aggregation" 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.

Furthermore, GNNs following this definition are regarded as Message-Passing-Neural-Network (MPNN). This designation stems from each node exchanging information with its direct neighbors in each layer. As a result, information during the processing of a graph is propagated by passing many messages across the graph; thus, these layers are also referred to as *message-passing* layers. As outlined in the introduction to this thesis, we will solely focus on GNNs utilizing the *message-passing* architecture. Therefore we will use the term GNN and MPNN interchangeably throughout this thesis. 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 instances of the definition, we will now introduce three distinct GNN architectures. Specifically, we will explore the Graph Attention Network (GAT) developed by Veličković et al. [2017], Graph Convolutional Network (GCN) proposed by Kipf and Welling [2017], and the Graph Isomorphism Network (GIN) introduced by Xu et al. [2019]. These architectures will serve as empirical baselines in Part II. Additionally, we will also elaborate on the reasons for this choice of models in Part II in Section 5.2. We listed the definitions of the *message-passing* layers of each model in **??**.

Table 1.: Overview of the construction of the *message-passing* layers and their respective learnable parameters by popular GNN architectures. A complete definition of the GAT architecture including the attention coefficient α_{vu} can be found in Definition 26 in the Appendix.

Model	Merge Function	Aggregation Function	Learnable Parameters
GAT	$f_{\text{merge}}^{(t)} = \sigma(\alpha_{vv} \cdot f^{(t)}(v) + f_{\text{agg}}^{(t)})$	$f_{\text{agg}}^{(t)} = \sum_{u \in \mathcal{N}(v)} \alpha_{vu} \cdot W^{(t)} \cdot f^{(t-1)}(u)$	$\alpha_{vu}, W^{(t)}$
GCN	$f_{\text{merge}}^{(t)} = \text{ReLU}\left(\frac{W^{(t)}}{1+d(v)} f^{(t-1)}(v) + f_{\text{agg}}^{(t)}\right)$	$f_{\text{agg}}^{(t)} = \sum_{u \in \mathcal{N}(v)} \frac{W^{(t)}}{\sqrt{(1+d(v)) \cdot (1+d(u))}} f^{(t-1)}(u)$	$W^{(t)}$
GIN	$f_{\text{merge}}^{(t)} = \text{MLP}^{(t)}\left((1+\epsilon^{(t)}) \cdot f^{(t-1)}(v) + f_{\text{agg}}^{(t)}\right)$	$f_{\text{agg}}^{(t)} = \sum_{u \in \mathcal{N}(v)} f^{(t-1)}(u)$	$\epsilon^{(t)}, \text{MLP}^{(t)}$

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. Although more sophisticated pooling operations exist, such as Set2Set developed by Vinyals et al. [2015], Xu et al. [2019] showed that given the correct configuration, the elementwise summation pooling function combined with a following multilayer perceptron suffices to create a GNN that is as expressive as the 1-WL algorithm in distinguishing non-isomorphism.

Rally small technical detail, but i dont thinnk the attention value fits into my defintion of GNNs

adapt caption!

4. Theoretical Connection

This section is the main part of our theoretical investigation of the two frameworks 1-WL+NN and GNN. We will present three 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. While the last theorem will take this a step further and demonstrate how powerful the 1-WL algorithm is by establishing a connection between 1-WL+NN and GNN for continuous functions.

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

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, the equivalence between both frameworks follows. Specifically, every function computed by 1-WL+NN working over any arbitrary, but finite $\mathcal{X} \subset \mathcal{G}$ is also computable by a GNN, and vice versa. As we move towards the empirical evaluation in Part II, it is evident that if we test a 1-WL+NN model on any of the benchmark datasets, it can achieve theoretically the same level of performance as a GNN model. 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 the 1-WL algorithm and in particular of collections of functions that are 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 proof and the following theorem that such a continuous version of the 1-WL algorithm exists.

We define the set of graphs with continuous labels 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 $\mathcal{X} \coloneqq X^{n \times n}$ throughout the next theorem.

Then the following theorem can be derived.

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.

The notion of a collection of functions capable of approximating any GNN function is defined as follows.

Definition 15 (GNN-Approximating)**.** Let $\mathcal{C}$ be a collection of permutation invariant functions from $\mathcal{X}$ to $\mathbb{R}$. We say $\mathcal{C}$ is GNN-Approximating if for all permutation-invariant functions $\mathcal{A}$ computed by a GNN, and for all $\epsilon \in \mathbb{R}$ with $\epsilon > 0$, there exists $h_{\mathcal{A},\epsilon} \in \mathcal{C}$ such that $\|\mathcal{A} - h_{\mathcal{A},\epsilon}\|_\infty \coloneqq \sup_{G \in \mathcal{X}} |\mathcal{A}(G) - h_{\mathcal{A},\epsilon}(G)| < \epsilon$

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 proof of Theorem 14 in the Appendix in **??**.

By putting all theorems into perspective, 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.

Redo, remove gapps and co!

4.1. Proof of Theorem 11: "GNN $\subseteq$ 1-WL+NN"

We will prove Theorem 12 by first introducing a set of lemmas, which will be leveraged in the proof of the theorem at the end of this section. The lemmas Lemmas 20 and 21, and the proof of Theorem 11 extend the results by Chen et al. [2019].

To begin, we prove an essential insight that for any pair of graphs indistinguishable by the 1-WL isomorphism test, the output of any 1-WL+NN model applied to both graphs is identical.

Lemma 16 (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 $G_1 \simeq_{1\text{WL}} G_2$ then $\mathcal{B}(G_1) = \mathcal{B}(G_2)$.

Proof. Assume the above. Let $\mathcal{B}$ be an arbitrary function over $\mathcal{X}$ computable by 1-WL+NN, then $\mathcal{B}$ is composed as follows: $\mathcal{B}(\cdot) = \mathsf{MLP} \circ f_{\text{enc}}\{\!\{\text{1-WL}(\cdot)(v) \mid v \in V(\cdot)\}\!\}$. Further, let $G_1, G_2 \in \mathcal{X}$ be arbitrary graphs with $G_1 \simeq_{1\text{WL}} G_2$, then by definition of the $\simeq_{1\text{WL}}$ relation we know that $\text{1-WL}(G_1) = \text{1-WL}(G_2)$. With this, the equivalence follows, since this implies the equivalence of the multiset of colors for both graphs. □

As a consequence of this lemma, we establish that every function computable by 1-WL+NN is also permutation invariant.

Corollary 17 (1-WL+NN Permuation Invariance)**.** Let $\mathcal{B}$ be a function over $\mathcal{X}$ computable by 1-WL+NN, then $\mathcal{B}$ is permutation invariant.

Proof. Assume the above. Let $G \in \mathcal{X}$ be an arbitrary graph and π a permutation of $V(G)$ the set of nodes, then we know that G is isomorph to $\pi \cdot G$. Since the 1-WL algorithm is sound, we know that $G \simeq \pi \cdot G$ implies $G \simeq_{1\text{WL}} \pi \cdot G$. Using Lemma 16, we can therefore conclude that: $\mathcal{B}(G) = \mathcal{B}(\pi \cdot G)$. □

With this property of 1-WL+NN functions, we can show the existence of a 1-WL-Discriminating collection of functions computable by 1-WL+NN with Lemma 18. It is necessary to prove this lemma as it forms the basis of Lemma 20.

Lemma 18. 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 will prove the lemma by giving a construction of such a collection. 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 $\mathcal{C}$ as follows:

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

where MLP_{id} is the identity function encoded as a multilayer perceptron that returns its input. Since every function $\mathcal{B}_c \in \mathcal{C}$ 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.

To prove that this collection is 1-WL-Discriminating, we need to show two properties: 1) Each function in the collection is permutation invariant, and 2) For each pair of graphs in $\mathcal{X}$ distinguishable by the 1-WL isomorphism test, there must exist a function in the collection that also distinguishes the pair.

For the first property, we already established in Corollary 17 that all 1-WL+NN functions are permutation invariant. For the second property, let $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{1\text{WL}} G_2$. 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 $G_1 \not\simeq_{1\text{WL}} G_2$, there exists a color $c \in \mathbb{N}$ such that $\text{hist}_{G_1,C_1}(c) \neq \text{hist}_{G_2,C_2}(c)$, such that $\mathcal{B}_c \in \mathcal{C}$ exists with $\mathcal{B}_c(G_1) \neq \mathcal{B}_c(G_2)$, satisfying the second property. □

The following Lemma 19 forms the basis in constructing 1-WL+NN computable functions in the subsequent proofs of the lemmas, as well as in the proof of the actual theorem. In detail, we will show that combining the output of several 1-WL+NN computable functions and further processing their concatenation of outputs with a multilayer perceptron is also 1-WL+NN computable.

Lemma 19 (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 $\mathsf{MLP}^\bullet$ an multilayer perceptron, then the function $\mathcal{B}$ composed of $\mathcal{B}(\cdot) \coloneqq \mathsf{MLP}^\bullet(h_1(\cdot), \dots, h_n(\cdot))$ 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 $\mathsf{MLP}_1, \dots, \mathsf{MLP}_n$ 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 MLP^* that takes in the output of f^* and simulates all $\mathsf{MLP}_1, \dots, \mathsf{MLP}_n$ simultaneously. Afterward, the given $\mathsf{MLP}^\bullet$ 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 Lemma 19.

$$G \xrightarrow{\mathsf{1\text{-}WL}} \begin{array}{c} M_G := \\ \{\!\{\mathsf{1\text{-}WL}(G)(v) \mid v \in V(G)\}\!\} \end{array} \xrightarrow{f^*} \begin{bmatrix} f_1(M_G) \\ \vdots \\ f_n(M_G) \end{bmatrix} \xrightarrow{\mathsf{MLP}^*} \text{MLP}^\bullet\Big(\begin{bmatrix} \text{MLP}_1(f_1(M_G)) \\ \vdots \\ \text{MLP}_n(f_n(M_G)) \end{bmatrix}\Big)$$

Figure 5.: The proof idea for Lemma 19, visualizing the functions f^* and MLP^* and how they work when applied on an input $G \in \mathcal{X}$. We denote here by M_G the multiset of the colors of the nodes of G after applying the 1-WL algorithm.

In the following two Lemmas 20 and 21, we will show that the indicator function $\mathbb{1}$ for the $\simeq_{1\text{WL}}$ relation on $\mathcal{X}$ is 1-WL+NN computable. We formally define this function for any pair of graphs $G_1, G_2 \in \mathcal{X}$ as follows:

$$\mathbb{1}_{G_1 \simeq_{1\text{WL}} G_2} = \begin{cases} 1, & \text{if } G_1 \simeq_{1\text{WL}} G_2 \\ 0, & \text{else} \end{cases} .$$

This function plays a crucial role in the proof of the theorem. We will first introduce an approximation of the inverse of this function in Lemma 20 and then use this approximation for the proof of Lemma 21 to construct a function $\varphi_{G_1}(G_2)$ that is equivalent to the indicator function $\mathbb{1}_{G_1 \simeq_{1\text{WL}} G_2}$.

Lemma 20. 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 on any input $G \in \mathcal{X} : h_{G^*}(G) = 0$, if and only if, $G \simeq_{1\text{WL}} G^*$.

Proof. Assume the above. Since $\mathcal{C}$ is 1-WL-Discriminating, we know that for any pair of graphs $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{1\text{WL}} G_2$, the function $h_{G_1,G_2} \in \mathcal{C}$ exists, that distinguishes the pair, such that $h_{G_1,G_2}(G_1) \neq h_{G_1,G_2}(G_2)$. We define the function $\overline{h}_{G_1,G_2}$ working over $\mathcal{X}$ for every such pair as follows:

$$\begin{aligned} \overline{h}_{G_1,G_2}(\cdot) &= |h_{G_1,G_2}(\cdot) - h_{G_1,G_2}(G_1)| \\ &= \max(h_{G_1,G_2}(\cdot) - h_{G_1,G_2}(G_1),\ 0) + \max(h_{G_1,G_2}(G_1) - h_{G_1,G_2}(\cdot),\ 0) \\ &= \text{ReLU}(h_{G_1,G_2}(\cdot) - h_{G_1,G_2}(G_1)) + \text{ReLU}(h_{G_1,G_2}(G_1) - h_{G_1,G_2}(\cdot)) \end{aligned} \tag{4.1}$$

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

$$h_{G_1}(\cdot) = \sum_{\substack{G_2 \in \mathcal{X} \\ G_1 \not\simeq_{1\text{WL}} G_2}} \overline{h}_{G_1,G_2}(\cdot). \tag{4.2}$$

Since $\mathcal{X}$ is finite, the sum is finite and therefore well-defined. Next, we will show that this construction fulfills the desired properties, by proving that for any input $G \in \mathcal{X} : h_{G_1}(G) = 0$, if and only if, $G \simeq_{1\text{WL}} G_1$. Note that G_1 is arbitrary but fixed. Let $G \in \mathcal{X}$ be an arbitrary input graph:

1. If $G_1 \simeq_{1\text{WL}} G$, then for every function $\overline{h}_{G_1,G_2}$ of the sum with $G_1 \not\simeq_{1\text{WL}} G_2$, we know, using Lemma 16, that $\overline{h}_{G_1,G_2}(G)$ is equal to $\overline{h}_{G_1,G_2}(G_1)$ which is by definition 0, such that $h_{G_1}(G) = 0$.
2. If $G_1 \not\simeq_{1\text{WL}} G$, then $\overline{h}_{G_1,G}(G)$ is a summand of the overall sum, and since $\overline{h}_{G_1,G}(G) > 0$, we can conclude $h_{G_1}(G) > 0$ due to the non-negativity of each $\overline{h}_{G_1,G_2}$ function.

Using Lemma 19, we can conclude that for any $G \in \mathcal{X}$, h_G is computable by 1-WL+NN, as we can encode Equation (4.2) via a multilayer perceptron where the constant $h_{G_1,G_2}(G_1)$ of Equation (4.1) will be the bias of the corresponding channel, such that this MLP exists.

It is crucial to note that in the special case where no pair of graphs within $\mathcal{X}$ is indistinguishable by the 1-WL isomorphism test from another, the function h_{G_1} is still defined. However, it sums over zero summands, resulting in $h_{G_1}(\cdot) = 0$. □

Thus, we have proved that the function h_G is 1-WL+NN computable for any graph $G \in \mathcal{X}$. The function h_G approximates the inverted indicator function for the fixed graph G by mapping graphs indistinguishable from G by the 1-WL algorithm to 0 while mapping every other graph to something strictly larger than 0. The following proof will use this property to construct the indicator function.

Lemma 21. 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 $h_{G^*} \in \mathcal{C}$ satisfying $h_{G^*}(G) = 0$, if and only if, $G \simeq_{1\text{WL}} G^*$, for all $G \in \mathcal{X}$. Then for every $G^* \in \mathcal{X}$, there exists a function φ_{G^*} computable by 1-WL+NN such that for all $G \in \mathcal{X}$: $\varphi_{G^*}(G) = \mathbb{1}_{G \simeq_{1\text{WL}} G^*}$.

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

$$\delta_{G^*} \coloneqq \frac{1}{2} \min_{\substack{G \in \mathcal{X} \\ G \not\simeq_{1\text{WL}} G^*}} |h_{G^*}(G)|.$$

The constant δ_{G^*} represents the minimum value to which the corresponding function $h_{G^*}(\cdot)$ maps a graph G that is distinguishable from G^* by the 1-WL isomorphism test, multiplied by the factor $\frac{1}{2}$. The specific value of this factor is arbitrary; the crucial aspect is that it remains less than 1, ensuring that the constant δ_{G^*} remains strictly smaller than the minimum value of $h_{G^*}(\cdot)$ for any graph G where $G \not\simeq_{1\text{WL}} G^*$.

It is important to note that in the special case where no pair of graphs within $\mathcal{X}$ is indistinguishable by the 1-WL isomorphism test from another, the constant δ_{G^*} is not well-defined. For these cases, we can set $\delta_{G^*} \coloneqq 1$ for all $G^* \in \mathcal{X}$.

We further introduce a so-called "bump" function $\psi_a(x)$ working from $\mathbb{R}$ to $\mathbb{R}$ paraemetrized by $a \in \mathbb{R}$ with $a > 0$ and defined as follows:

$$\begin{aligned} \psi_a(x) &\coloneqq \max(\frac{x}{a} - 1,\ 0) + \max(\frac{x}{a} + 1,\ 0) - 2 \cdot \max(\frac{x}{a},\ 0) \\ &= \text{ReLU}(\frac{x}{a} - 1) + \text{ReLU}(\frac{x}{a} + 1) - 2 \cdot \text{ReLU}(\frac{x}{a}) \end{aligned} \tag{4.3}$$

The interesting property of ψ_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 and the constant δ_{G^*} to define for every graph $G^* \in \mathcal{X}$ the function φ_{G^*} that is equivalent to the indicator function as follows:

$$\varphi_{G^*}(\cdot) \coloneqq \psi_{\delta_{G^*}}(h_{G^*}(\cdot)).$$

We will prove the correctness of this construction by showing that for a fixed graph G^* the following condition holds: $\forall G \in \mathcal{X} : \varphi_{G^*}(G) = 1$, if and only if, $G \simeq_{1\text{WL}} G^*$. For this lets consider two cases:

1. If $G \simeq_{1\text{WL}} G^*$, then $h_{G^*}(G) = 0$ resulting in $\varphi_{G^*}(G) = \psi_{\delta_{G^*}}(0) = 1$.
2. If $G \not\simeq_{1\text{WL}} G^*$ 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 φ_{G^*} using Equation (4.3) via a multilayer perceptron, where δ_{G^*} is a constant, such that this MLP exists. With Lemma 19 we can therefore conclude that φ_{G^*} is computable by 1-WL+NN for every graph $G^* \in \mathcal{X}$. □

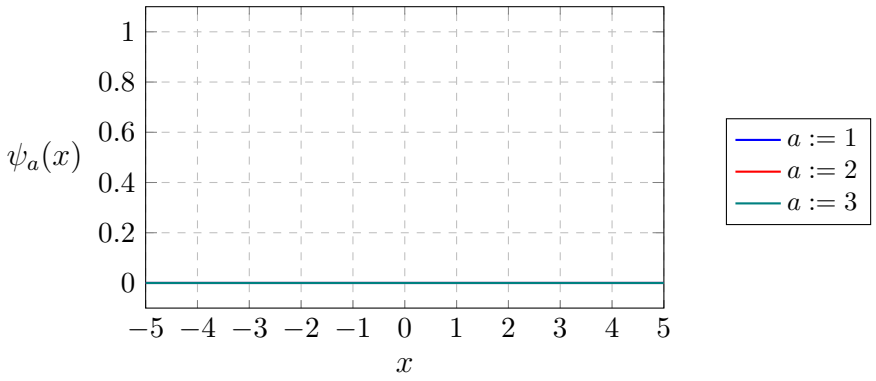


Figure 6.: Illustration of the so-called "bump" function $\psi_a(x)$ used in the proof of Lemma 21 with different exemplary values for a.

We can now leverage all our lemmas to prove the overall theorem that any function $\mathcal{A}$ computable by a GNN can also be computed by 1-WL+NN.

Proof of Theorem 11. Let $\mathcal{A}$ be a function that works over $\mathcal{X}$ to $\mathbb{R}$ computed by a GNN model. We will prove that $\mathcal{A}$ is 1-WL+NN computable by decomposing the function and then argue that the decomposition is computable by a 1-WL+NN model. For this let $G \in \mathcal{X}$ be an arbitrary input graph, we can decompose $\mathcal{A}(G)$ as follows:

$$\mathcal{A}(G) = \Big(\frac{1}{|\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G)|} \sum_{G^* \in \mathcal{X}} \mathbb{1}_{G \simeq_{1\mathrm{WL}} G^*} \Big) \cdot \mathcal{A}(G) \tag{4.4}$$

$$= \sum_{G^* \in \mathcal{X}} \frac{1}{|\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G)|} \cdot \mathcal{A}(G) \cdot \mathbb{1}_{G \simeq_{1\mathrm{WL}} G^*} \tag{4.5}$$

$$= \sum_{G^* \in \mathcal{X}} \frac{1}{|\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G^*)|} \cdot \mathcal{A}(G^*) \cdot \mathbb{1}_{G \simeq_{1\mathrm{WL}} G^*} \tag{4.6}$$

$$= \sum_{G^* \in \mathcal{X}} \frac{\mathcal{A}(G^*)}{|\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G^*)|} \cdot \varphi_{G^*}(G) \tag{4.7}$$

where $\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G)$ denotes the set of all graphs that are equivalent to G according to the $\simeq_{1\mathrm{WL}}$ relation. We explain each equation step by step:

Equation (4.4): Multiplying $\mathcal{A}(G)$ by the factor in the parentheses is correct because it is equal to 1. This is because the sum "counts" the number of graphs $G^* \in \mathcal{X}$ that are indistinguishable from the input graph G by the 1-WL isomorphism test, and then the count is divided by the number of graphs in the equivalence class of G, which is the same as the count.

Equation (4.5): We can move both the factor $\mathcal{A}(G)$ and $\frac{1}{|\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G)|}$ into the sum by using the distributive property of the space $\mathbb{R}$.

Equation (4.6): Due to the output of the indicator function $\mathbb{1}$ being either 0 or 1, we can infer that the inner product of each summand can only be nonzero if G^* is indistinguishable by the 1-WL isomorphism test from G. This implies that both are in the same equivalence class in these cases, such that $|\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G)|$ is equal to $|\mathcal{X}/\!\simeq_{1\mathrm{WL}}(G^*)|$.

Additionally, since GNNs are, at most, as good as the 1-WL algorithm in distinguishing pairs of non-isomorphic graphs (Morris et al. [2019], Xu et al. [2019]), we can use the fact that for every graph $G^* \in \mathcal{X}$: if $G^* \simeq_{1\mathrm{WL}} G$, then $\mathcal{A}(G^*) = \mathcal{A}(G)$. Using the same reasoning with the indicator function, we can replace the term $\mathcal{A}(G)$ by $\mathcal{A}(G^*)$.

Equation (4.7): Utilizing Lemma 21, we can replace the indicator function with $\varphi_{G^*}(G)$.

In conclusion, we have decomposed the GNN function $\mathcal{A}(G)$ such that the only factors that depend on the input graph G are the functions φ_{G^*}, which take G as input. This observation implies that all other factors are constants. Consequently, we can reason that the entire decomposition can be computed by a multilayer perceptron with a single layer, which takes the output of all φ_{G^*} for all $G^* \in \mathcal{X}$, applied to the input graph G. The multilayer perceptron then multiplies each of these values with the constant $\frac{\mathcal{A}(G^*)}{|\mathcal{X}/\simeq_{1\mathrm{WL}}(G^*)|}$ and takes the sum. The existence of such a multilayer perceptron is evident, and when combined with Lemma 19, we can assert that this composition is 1-WL+NN computable.

Important to note, we can only do this since $\mathcal{X}$ is finite, making the overall sum finite and the cardinality of $\mathcal{X}/\simeq_{1\mathrm{WL}}(G^*)$ well-defined for all graphs. □

4.2. Proof of Theorem 12: "1-WL+NN $\subseteq$ GNN"

In this section, we will prove the converse direction. Similar to the previous subsection, we will begin by introducing a set of lemmas that will play a crucial role in proving Theorem 12.

We start by showing the existence of a collection of functions computable by GNNs that is 1-WL-Discriminating. For the proof, we will devise message-passing layers for a GNN that effectively compute a single iteration of the 1-WL algorithm per layer. Afterward, we show that with a proper choice of the Readout function, we construct a collection of GNN functions that is 1-WL-Discriminating. Although prior works by Morris et al. [2019] and Xu et al. [2019] have already demonstrated how to construct message-passing layers to compute a single iteration of the 1-WL algorithm per layer, we include our own construction with our notation in the proof for two crucial reasons. Firstly, it ensures the completeness of our proof without assuming major parts. Secondly, and most importantly, it effectively highlights the remarkable similarities and key distinctions between the 1-WL algorithm and GNNs in general.

Lemma 22 (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.

Proof. Due to $\mathcal{X}$ being finite, we define the constants n, m, k as follows:

$$n := \max_{G \in \mathcal{X}} |V(G)|, \quad m := \sum_{G \in \mathcal{X}} |V(G)|, \quad \text{and} \quad k := 1 + \max_{\substack{G \in \mathcal{X} \\ v \in V(G)}} |l_G(v)|,$$

such that n is the maximum number of nodes of any graph in $\mathcal{X}$, m is the total number of nodes of the set $\mathcal{X}$, and k is the largest label of any node of a graph in $\mathcal{X}$ plus 1.

We will utilize these constants to construct the collection of functions $\mathcal{C} := \{\mathcal{A}_c \mid c \in \mathbb{N}\}$ that is 1-WL-Discriminating. For the remainder of this proof, we first describe the construction of an arbitrary $\mathcal{A}_c \in \mathcal{C}$ and afterward, prove that the collection $\mathcal{C}$ is 1-WL-Discriminating.

Each $\mathcal{A}_c$ consists of n message-passing layers. We define the input layer $f^{(0)}(v) := v$ as the identity functions such that there is no preprocessing of the node labels. Further, we define

every other layer t with $1 \leq t \leq n$ as follows:

$$f^{(t)}(v) \coloneqq f^{(t)}_{\text{merge}}(f^{(t-1)}(v),\ \{\!\{f^{(t-1)}(u) \mid u \in \mathcal{N}(v)\}\!\}).$$

Here $f^{(t)}_{\text{merge}}$ is an injective function that maps its input into its codomain:

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

This function exists due to the finiteness of $\mathcal{X}$. We can upper bound the cardinality of its domain, the number of unique tuples, by the total number of nodes in $\mathcal{X}$, which is m, and since the cardinality of its codomain is exactly m, we can conclude the existence of the function.

Next, we will define the Readout function of $\mathcal{A}_c$ to be the function that returns the number of nodes colored as c in the coloring of $f^{(n)}$.

By leveraging the results of *theorem 3* from the work of Xu et al. [2019], we can infer that each layer of each $\mathcal{A}_c$ computes a single iteration of the 1-WL algorithm. This observation makes sense as the update equation for each layer injectively maps each tuple to a previously unused color, similar to how the Relabel function of the 1-WL algorithm works. Moreover, since the 1-WL algorithm terminates on any graph $G \in \mathcal{X}$ after at most $|V(G)| \leq n$ iterations, the coloring computed by the layers of each $\mathcal{A}_c$ effectively performs n iterations of the 1-WL algorithm when applied to any graph $G \in \mathcal{X}$. Due to the convergence behavior of the 1-WL algorithm, these additional iterations do not increase the expressiveness of the colorings computed by each $\mathcal{A}_c$, such we can conclude for any graph $G \in \mathcal{X}$:

$$\forall c \in \mathbb{N} : |\{v \in V(G) \mid f^{(n)}(v) = c\}| = |\{v \in V(G) \mid \text{1-WL}(G)(v) = c\}|,$$

which states that the colorings are equal for a bijection $\phi : \mathbb{N} \to \mathbb{N}$, such that we can infer that they are equally expressive for distinguishing non-isomorphism.

To prove that the collection $\mathcal{C}$ is 1-WL-Discriminating, we need to show two properties: 1) Each function in the collection is permutation invariant, and 2) For each pair of graphs in $\mathcal{X}$ distinguishable by the 1-WL isomorphism test, there must exist a function in the collection that also distinguishes the pair.

For the first property, by Definition 10 of GNNs, all functions computed by GNNs are permutation-invariant. Regarding the second property, consider $G_1, G_2 \in \mathcal{X}$ with $G_1 \not\simeq_{\text{1WL}} G_2$. Let C_1 and C_2 represent the final colorings computed by the 1-WL algorithm when applied to G_1 and G_2, respectively. Since $G_1 \not\simeq_{\text{1WL}} G_2$, there exists a color $c \in \mathbb{N}$ such that $\text{hist}_{G_1,C_1}(c) \neq \text{hist}_{G_2,C_2}(c)$. Since, we know that each $\mathcal{A}_c$ computes equally expressive colorings of G_1 and G_2, we know that there exists a $c' \in \mathbb{N}$, such that $\mathcal{A}_{c'}(G_1) \neq \mathcal{A}_{c'}(G_2)$. □

Similar to the proof in the previous subsection, we will use Lemma 23 to introduce the ability to construct GNNs that take in as input multiple GNNs and then apply a multilayer perceptron to the combined output. This insight is leveraged in the following two corollaries in the proof, as well as in the final proof.

Lemma 23 (GNN Composition)**.** Let $\mathcal{C}$ be a collection of functions computable by GNNs. Further, let $\mathcal{A}_1, \ldots, \mathcal{A}_n \in \mathcal{C}$ and $\text{MLP}^\bullet$ a suitable multilayer perceptron, then the function $\hat{\mathcal{A}}(\cdot) \coloneqq \text{MLP}(\mathcal{A}_1(\cdot), \ldots, \mathcal{A}_n(\cdot))$ is also computable by a GNN.

Proof. Before we begin the proof, we briefly introduce two notations. 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 $f^{(t)}_{\text{merge},i}$ and $f^{(t)}_{\text{agg},i}$. Similarly, we denote the $\mathsf{Readout}$ function as $\mathsf{Readout}_i$ and the input function of $\mathcal{A}_i$ as $f^{(0)}_i$.

We will prove the lemma by giving a construction of a GNN model computing $\hat{\mathcal{A}}$. For the ease of readability and to reduce the complexity of the subsequent construction, we assume that for all $\mathcal{A}_i$ its functions $f^{(t)}_{\text{merge},i}, f^{(t)}_{\text{agg},i}$ and $\mathsf{Readout}_i$ map into the one-dimensional space $\mathbb{R}$ for all layers t. With this assumption, we avoid the need for a formal notation of the number of dimensions each of these functions map to.

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

$$\forall v \in V(G) : \ \hat{f}^{(0)}(v) \coloneqq \begin{bmatrix} f^{(0)}_1(v) \\ \vdots \\ f^{(0)}_n(v) \end{bmatrix},$$

and each other layer $0 < t \leq T$ utilizing the merge $\hat{f}^{(t)}_{\text{merge}}$ and aggregation $\hat{f}^{(t)}_{\text{agg}}$ functions as constructed in the following:

$$\hat{f}^{(t)}_{\text{merge}}(\hat{f}^{(t-1)}(v),\ Agg) \coloneqq \begin{bmatrix} f^{(t)}_{\text{merge},1}(\hat{f}^{(t-1)}(v)[1],\ Agg[1]) \\ \vdots \\ f^{(t)}_{\text{merge},n}(\hat{f}^{(t-1)}(v)[n],\ Agg[n]) \end{bmatrix}, \quad \text{and}$$

$$\hat{f}^{(t)}_{\text{agg}}(\{\!\{\hat{f}^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}\!\}) \coloneqq \begin{bmatrix} f^{(t)}_{\text{agg},1}(\{\!\{\hat{f}^{(t-1)}(w)[1] \mid w \in \mathcal{N}(v)\}\!\}) \\ \vdots \\ f^{(t)}_{\text{agg},n}(\{\!\{\hat{f}^{(t-1)}(w)[n] \mid w \in \mathcal{N}(v)\}\!\}) \end{bmatrix}.$$

Note that, not all $\mathcal{A}_i$ will be comprised of T layers, such that for these cases the functions $f^{(t)}_{\text{merge},i}$ and $f^{(t)}_{\text{agg},i}$ will not be defined for all $t \in [T]$. In these cases, we define the functions as follows:

$$f^{(t)}_{\text{merge},i}(\hat{f}^{(t-1)}(v),\ Agg) \coloneqq \hat{f}^{(t-1)}(v), \quad \text{and}$$
$$f^{(t)}_{\text{agg},i}(\{\!\{\hat{f}^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}\!\}) \coloneqq 0.$$

This definition of $f^{(t)}_{\text{merge},i}$ and $f^{(t)}_{\text{agg},i}$ results in the fact that the representation computed in the last layer of $\mathcal{A}_i$ is forwarded to the last layer T of $\hat{\mathcal{A}}$. Finally, we construct the $\mathsf{Readout}$ function of $\hat{\mathcal{A}}$ as follows:

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

With this, the proof concludes. Note that this proof can easily be extended to work without the assumption of each function mapping into a one-dimensional space. □

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 22, 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 23, we have demonstrated that the composition of multiple GNNs and a multilayer perceptron remains computable by a single GNN. Consequently, we can use the same proofs of Lemmas 20 and 21 to derive the Corollaries 24 and 25.

Corollary 24. Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by GNNs 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 GNN, such that on any input $G \in \mathcal{X} : h_{G^*}(G) = 0$, if and only if, $G \simeq_{\text{1WL}} G^*$.

Corollary 25. Let $\mathcal{C}$ be a collection of functions from $\mathcal{X}$ to $\mathbb{R}$ computable by GNNs such that for all $G^* \in \mathcal{X}$, there exists $h_{G^*} \in \mathcal{C}$ satisfying $h_{G^*}(G) = 0$, if and only if, $G \simeq_{\text{1WL}} G^*$, for all $G \in \mathcal{X}$. Then for every $G^* \in \mathcal{X}$, there exists a function φ_{G^*} computable by GNNs such that for all $G \in \mathcal{X}$: $\varphi_{G^*}(G) = \mathbb{1}_{G \simeq_{\text{1WL}} G^*}$.

In conclusion, the corollaries establish the computability of the indicator function $\mathbb{1}_{G_1 \simeq_{\text{1WL}} G_2}$ over the set $\mathcal{X}$ by a GNN. Building upon these results, we can now utilize all our lemmas to prove the theorem, which states that any function $\mathcal{B}$ computable by a 1-WL+NN can also be computed by a GNN.

Proof of Theorem 12. Let $\mathcal{B}$ be a function that works over $\mathcal{X}$ to $\mathbb{R}$ computed by a 1-WL+NN model. We will prove that $\mathcal{B}$ is GNN computable by decomposing the function and then argue that the decomposition is computable by a GNN model. For this let $G \in \mathcal{X}$ be an arbitrary input graph, we can decompose $\mathcal{B}(G)$ as follows:

$$
\begin{aligned}
\mathcal{B}(G) &= \Big(\frac{1}{|\mathcal{X}/\!\simeq_{\text{1WL}}(G)|} \sum_{G^* \in \mathcal{X}} \mathbb{1}_{G \simeq_{\text{1WL}} G^*} \Big) \cdot \mathcal{B}(G) \\
&= \sum_{G^* \in \mathcal{X}} \frac{1}{|\mathcal{X}/\!\simeq_{\text{1WL}}(G)|} \cdot \mathcal{B}(G) \cdot \mathbb{1}_{G \simeq_{\text{1WL}} G^*} \\
&= \sum_{G^* \in \mathcal{X}} \frac{1}{|\mathcal{X}/\!\simeq_{\text{1WL}}(G^*)|} \cdot \mathcal{B}(G^*) \cdot \mathbb{1}_{G \simeq_{\text{1WL}} G^*} \qquad (4.8) \\
&= \sum_{G^* \in \mathcal{X}} \frac{\mathcal{B}(G^*)}{|\mathcal{X}/\!\simeq_{\text{1WL}}(G^*)|} \cdot \varphi_{G^*}(G) \qquad (4.9)
\end{aligned}
$$

where $\mathcal{X}/\!\simeq_{\text{1WL}}(G)$ denotes the set of all graphs that are equivalent to G according to the $\simeq_{\text{1WL}}$ relation. Since the decomposition is very similar to the one present in the proof of Theorem 11, we will only provide reasoning for the correctness of Equations (4.8) and (4.9). For all other equations, refer to the explanation provided in the proof of Theorem 11

Equation (4.8): Due to the output of the indicator function $\mathbb{1}$ being either 0 or 1, we can infer that the inner product of each summand can only be nonzero if G^* is indistinguishable by the 1-WL isomorphism test from G. Using Lemma 16, we know that for every graph $G^* \in \mathcal{X}$: if $G^* \simeq_{\text{1WL}} G$, then $\mathcal{B}(G^*) = \mathcal{B}(G)$.

Equation (4.9): Utilizing Corollary 25, we can replace the indicator function with $\varphi_{G^*}(G)$.

In conclusion, we have decomposed the GNN function $\mathcal{B}(G)$ such that the only factors that depend on the input graph G are the functions φ_{G^*}, which take G as input. This observation implies that all other factors are constants. Consequently, we can reason that the entire decomposition can be computed by a multilayer perceptron with a single layer, which takes

the output of all φ_{G^*} for all $G^* \in \mathcal{X}$, applied to the input graph G. The multilayer perceptron then multiplies each of these values with the constant $\frac{\mathcal{B}(G^*)}{|\mathcal{X}/\simeq_{1\mathrm{WL}}(G^*)|}$ and takes the sum. The existence of such a multilayer perceptron is evident, and when combined with Lemma 19, we can assert that this composition is $\mathsf{1\text{-}WL{+}NN}$ computable.

Important to note, we can only do this since $\mathcal{X}$ is finite, making the overall sum finite and the cardinality of $\mathcal{X}/\simeq_{1\mathrm{WL}}(G^*)$ well-defined for all graphs. □

Part II.

Empirical Testing

The empirical part of this thesis is structured into three sections. Firstly, we delve into the methodology used for conducting our experiments, covering the selection of models and datasets, the testing procedure, and the choice of hyperparameters. Subsequently, Section 6 will comprehensively explore the results obtained from the testing and examine various properties of both model types in detail. Finally, Section 7 concludes this thesis with a discussion that consolidates all insights, highlights any identified issues, and provides recommendations for future research directions.

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 for 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 ($\log P - \mathrm{SA} - \mathrm{cycle}$).

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	∅ Number of nodes	∅ Number of edges	Node labels	Edge labels	Taxonomy Category
Classification	Enzymes	600	6	32.6	62.1	✓	✗	1
	Imdb-Binary	1 000	2	19.8	96.5	✗	✗	3
	Mutag	188	2	17.9	19.8	✓	✗	2
	Nci1	4 110	2	29.9	32.3	✓	✗	3
	Proteins	1 113	2	39.1	72.8	✓	✗	2
	Reddit-Binary	2 000	2	429.6	497.8	✗	✗	3
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 Definition 28 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
Normalized 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 a 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 analyses 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. Further, we formally defined each of these architectures in **??** in Part I.

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 as these datasets allow for better scalability due to their smaller size than the regression datasets. Therefore, the thesis will mainly investigate these datasets.

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 by 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. 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 fixed 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 trustworthiness of our empirical results and gain valuable insights into the optimal configuration of 1-WL+NN models, we conducted a thorough hyperparameter optimization for each type of model, both GNN and 1-WL+NN, and performed this optimization individually for each dataset.

In order 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 for, can be found in Table B.2 for the 1-WL+NN models used in classification tasks, Table B.5 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 results achieved by 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 doing so, the number of colors used by the 1-WL algorithm remained contained, both in terms of the number of total unique colors as well as the distance to another. 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[6] and its extension PyTorch Geometric[7]. 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 6 754 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 2 114 hours (over 88 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 requires 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-ε model proposed by Morris et al. [2020] and Morris et al. [2022] in Table 6. The GINE-ε 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-ε 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

[6] 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

[7] An open-source library that extends PyTorch and allows for easy development and training of graph neural networks. https://pytorch-geometric.readthedocs.io

[8] The results of GINE-ε on Alchemy and Zinc are adopted from Morris et al. [2020].

[9] The results of GINE-ε on Alchemy(10k) is adopted from Morris et al. [2022].

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					
		Enzymes	Imdb-Binary	Mutag	Nci1	Proteins	Reddit-Binary
1-WL+NN	Max	16.7 ±4.2	52.0 ±5.3	73.8 ±12.4	58.6 ±3.3	62.9 ±4.9	69.2 ±4.0
	Mean	18.2 ±4.8	59.4 ±5.8	77.1 ±11.5	64.0 ±3.3	60.9 ±4.5	66.1 ±3.2
	Sum	18.0 ±6.2	57.5 ±5.1	66.8 ±13.9	56.9 ±3.8	65.6 ±4.8	73.0 ±5.1
	Embedding-Max	41.9 ±7.5	69.4 ±4.9	81.1 ±11.2	82.7 ±2.0	**75.2** ±3.9	71.8 ±3.2
	Embedding-Mean	45.8 ±6.8	**72.4** ±4.1	84.1 ±9.1	83.1 ±1.9	72.7 ±4.6	75.4 ±2.8
	Embedding-Sum	**48.3** ±8.1	72.0 ±3.8	**85.1** ±8.6	**83.6** ±2.2	75.2 ±4.5	**78.4** ±2.7
Graph Neural Networks	GAT:Max	31.2 ±6.0	70.7 ±4.8	71.1 ±12.2	58.0 ±4.2	72.5 ±5.1	77.7 ±5.0
	GAT:Mean	28.9 ±5.9	70.9 ±3.7	74.8 ±9.1	66.1 ±2.8	64.9 ±6.4	70.0 ±6.9
	GAT:Sum	**34.4** ±7.0	72.2 ±4.5	82.1 ±11.2	69.8 ±2.6	73.4 ±3.9	78.1 ±4.2
	GCN:Max	33.1 ±7.5	73.5 ±4.1	74.5 ±11.3	61.1 ±3.6	69.8 ±5.9	74.3 ±4.5
	GCN:Mean	29.9 ±5.7	**74.7** ±3.8	75.0 ±10.4	68.9 ±2.4	70.9 ±5.2	82.3 ±3.3
	GCN:Sum	31.7 ±7.2	73.0 ±4.4	81.5 ±10.3	70.4 ±2.1	73.9 ±4.0	**86.9** ±3.2
	GIN:Max	29.2 ±6.2	70.8 ±4.7	77.3 ±10.7	**79.9** ±2.2	**74.3** ±5.1	76.1 ±3.6
	GIN:Mean	31.7 ±6.7	71.1 ±5.4	82.4 ±9.8	71.3 ±2.2	72.0 ±4.0	73.5 ±3.8
	GIN:Sum	28.9 ±8.7	69.5 ±4.8	**84.6** ±8.7	70.8 ±2.3	73.2 ±4.3	74.0 ±4.3

Table 6.: Overview of the mean absolute error and the standard deviation (logMAE) on large-scale (multi-target) molecular regression tasks.

	Method	Dataset			
		Alchemy	Alchemy (10k)	Zinc	Zinc (10k)
GNN	GINE-ε	0.103 ±0.001 -2.956 ±0.029[8]	0.180 ±0.006 -1.958 ±0.047[9]	-	0.084 ±0.004[8]
	GIN:Max	0.604 ±0.004 -0.578 ±0.009	0.353 ±0.003 -1.228 ±0.006	0.124 ±0.004	0.427 ±0.009
	GIN:Mean	0.643 ±0.014 -0.505 ±0.021	0.314 ±0.004 -1.469 ±0.044	0.110 ±0.005	0.339 ±0.032
	GIN:Sum	**0.523** ±0.016 -0.705 ±0.035	**0.282** ±0.002 -1.890 ±0.031	**0.104** ±0.005	**0.298** ±0.034
1-WL+NN	Embedding-Max	0.648 ±0.003 -0.511 ±0.009	0.409 ±0.003 -1.023 ±0.009	0.382 ±0.005	0.659 ±0.007
	Embedding-Mean	0.617 ±0.003 -0.564 ±0.005	0.355 ±0.004 -1.269 ±0.020	**0.229** ±0.003	0.484 ±0.009
	Embedding-Sum	**0.600** ±0.004 -0.625 ±0.032	**0.305** ±0.001 -1.740 ±0.042	0.326 ±0.014	**0.465** ±0.009

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-ε 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 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] and 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 differences in their learning behaviors, the ability of GNN models to approximate 1-WL+NN models, the learned pooled graph representations of each model type, the impact of GNN models performance by preprocessing the data, and finally the effect of large datasets on the performance of 1-WL+NN models.

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 the overall distance between each used color.

From a theoretical perspective, fixing the number of 1-WL iterations in a 1-WL+NN model only limits its expressiveness. For instance, let $k \in \mathbb{N}$ now be fixed. If the standard 1-WL algorithm converges on an input graph G after $k' < k$ iterations, the coloring computed after k 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 k 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 algorithm types. Additionally, we included the mean number of iterations the 1-WL algorithm runs when being applied on each graph of a dataset. For comparison, we also included the number of 1-WL iterations used by the best-performing model utilizing the parametrized version of the 1-WL algorithm.

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.

Algorithm Type		**Dataset**					
		ENZYMES	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Standard	Test Accuracy:	34.2 ±6.7	67.0 ±4.3	76.3 ±9.4	78.9 ±2.1	71.4 ±4.9	70.9 ±3.8
	∅ 1-WL Iterations:	3.0 ±1.7	1.1 ±0.7	4.2 ±1.6	3.9 ±1.7	3.0 ±1.7	4.1 ±0.9
Parametrized	Test Accuracy:	48.3 ±8.1	72.4 ±4.1	85.1 ±8.6	83.6 ±2.2	75.2 ±3.9	78.4 ±2.7
	# 1-WL Iterations:	1	1	1	3	1	1

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

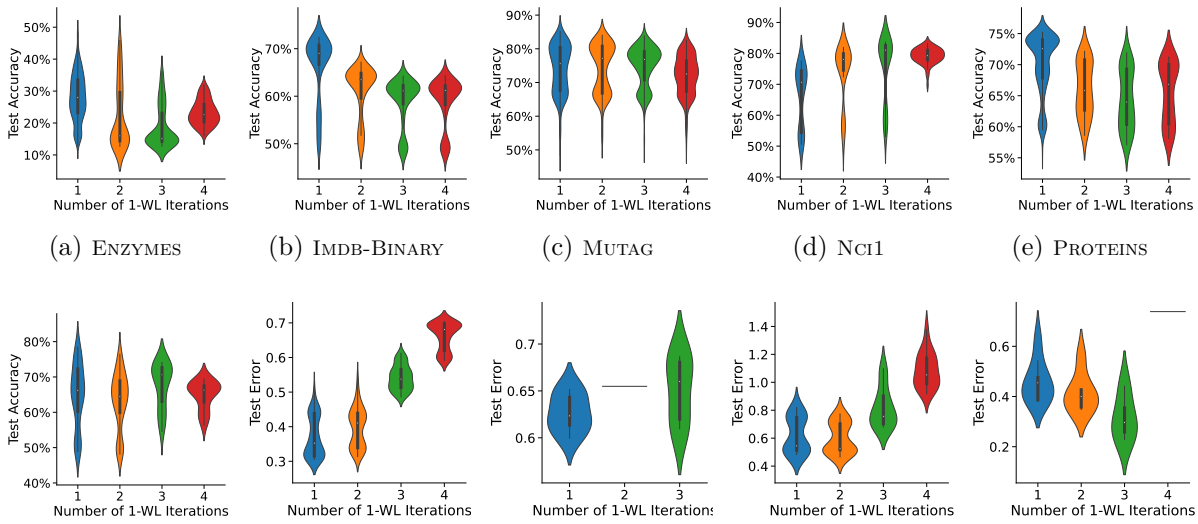
grouped by the number of 1-WL iterations for each dataset in a violin graph. Note that both classification and regression datasets are included in this figure. Depending on the dataset type, the y-axis represents either classification accuracy or test error, where higher accuracies imply better performance 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 the NCI1 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 the first iteration of the 1-WL alogirhtm is efficiently 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



(a) ENZYMES (b) IMDB-BINARY (c) MUTAG (d) NCI1 (e) PROTEINS

(f) REDDIT-BINARY (g) ALCHEMY (10K) (h) ALCHEMY (i) ZINC (10K) (j) ZINC

Figure 7.: Impact on performance by the number of itertations of the 1-WL alogirhtm.

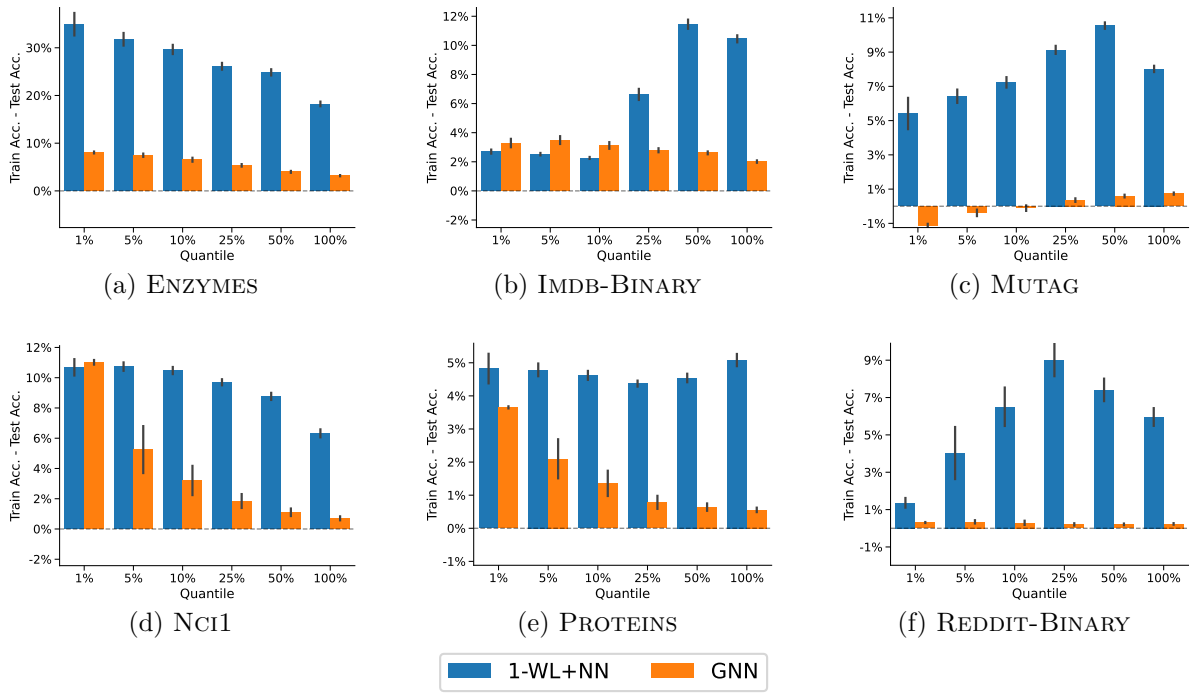
expressive, also tend to rely primarily on node degree information for computation. We will explore this question in the following section.

6.2. Difference in Learning Behavior between GNN and 1-WL+NN Models

The learning behavior of 1-WL+NN models presents a unique challenge due to the expressiveness of the 1-WL algorithm, which directly impacts their performance. Compared to GNN models, 1-WL+NN models exhibit a significant discrepancy between their training and test performance. While it is common to observe higher training performance than testing performance in various machine learning applications, this discrepancy is particularly pronounced in our evaluation of the 1-WL+NN models.

Figure 8 visually represents the difference in performance between the training and testing performance for both 1-WL+NN and GNN models across all classification datasets. Different quantiles from the best-performing models were considered, ranging from the top 1 % to the top 100 %. Each bar in the graph represents the mean difference between training accuracy and test accuracy for the respective quantile of runs. 1-WL+NN models are depicted in blue, while GNN models are represented in orange. In comparison, shorter or negative bars indicate better generalization where the performance on the training set aligns more closely with the testing set. The black line in each bar represents the standard error.

Upon analyzing Figure 8, it reveals that 1-WL+NN models exhibit more significant overfitting than GNN models, especially on datasets such as Enzymes, Mutag, Proteins, and Reddit-Binary. For instance, in the case of Enzymes, the top 1 % of best-performing 1-WL+NN



(a) Enzymes (b) Imdb-Binary (c) Mutag

(d) Nci1 (e) Proteins (f) Reddit-Binary

Figure 8.: Mean difference of the classification accuracies of the training and testing set for each dataset. In detail, we grouped by different quantiles of the best performing models and the type of the model.

models display, on average, a 27 % higher training accuracy than their test accuracy, highlighting the extreme extent of overfitting in 1-WL+NN models.

While GNN models also experience some overfitting, primarily observed in the NCI1 dataset, it is not as prevalent across all datasets and not as dramatic as in the case of 1-WL+NN models. Moreover, in datasets like MUTAG or REDDIT-BINARY, GNN models demonstrate superior generalization capability. In these cases, the mean difference in performance is close to 0 %, and in the case of Mutag, it is even negative, which indicates that the tested GNN models were able to learn general patterns in these datasets rather than simply memorizing the data.

However, this raises the question as to why 1-WL+NN models exhibit such dramatic overfitting. We hypothesize that the expressiveness of the 1-WL algorithm leads to the computation of highly detailed colorings of the 1-WL+NN's input graphs, such that the models simply memorizes these.

To gain insight into the algorithm's expressiveness, we calculated the ratio of unique colors to the total number of possible colors for each classification dataset. For example, the ENZYMES dataset consists of 600 graphs with a total of 19 580 nodes. After a single iteration of the 1-WL algorithm, all colorings consist of only 231 unique colors, accounting for less than 0.01 % of the total number of possible unique colors (the total number of nodes). However, after two iterations, this number has already increased to 10 416, comprising 53.2 % of the total available colors. This example demonstrates the significant expressiveness of the colorings after just a few iterations.

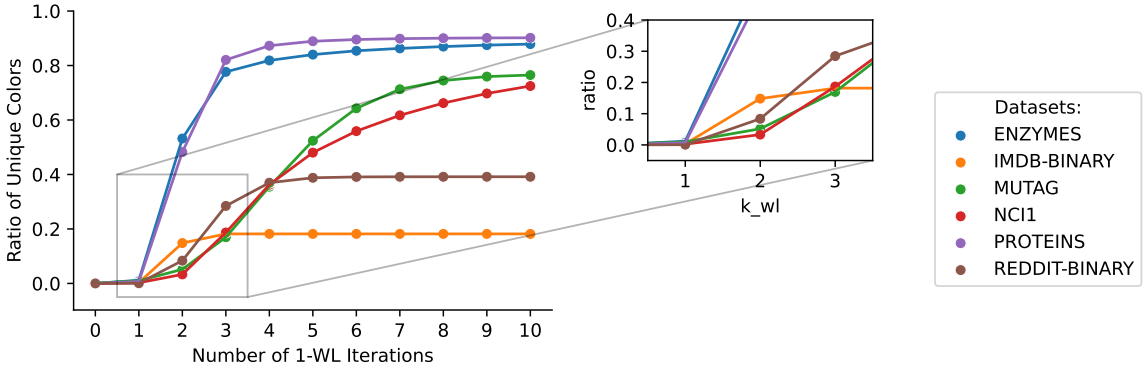


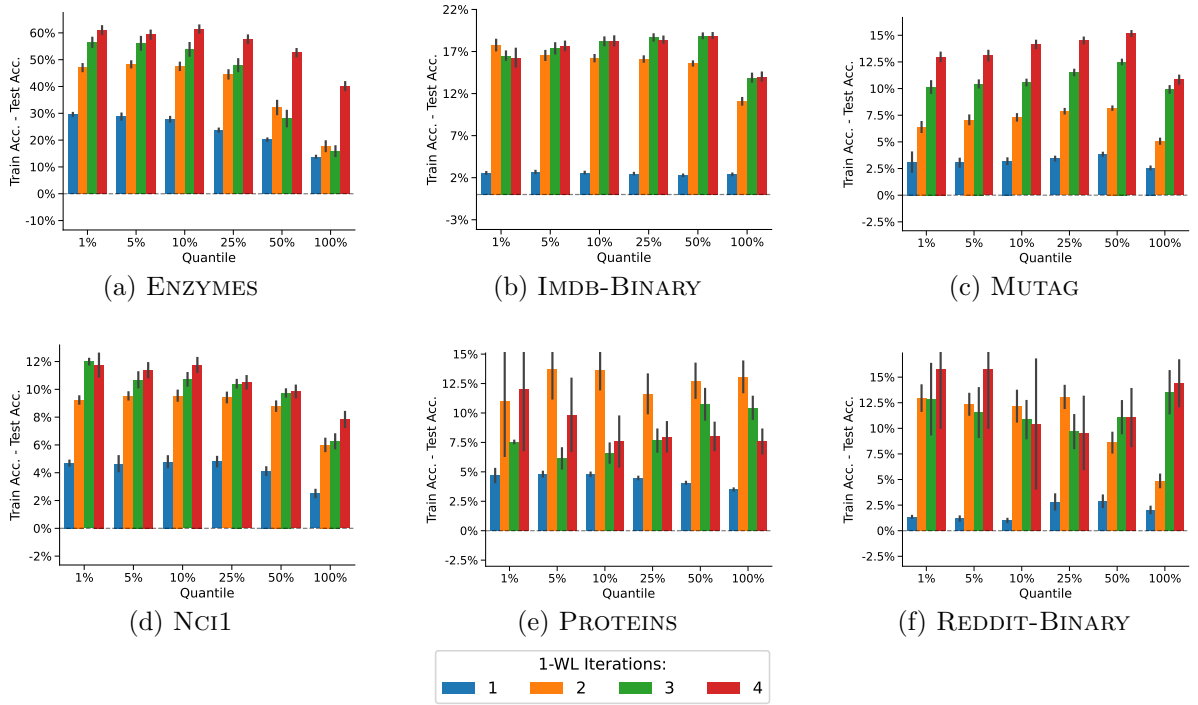
Figure 9.: Overview of the ratio of unique colors used by the 1-WL algorithm when applied for a specified number of iterations for each dataset. The ratios are computed using each dataset's total number of nodes as the maximum number of unique colors that could appear in all colorings.

Figure 9 illustrates the ratio of unique colors used compared to the total possible number of unique colors (total number of nodes in each dataset). As expected due to the convergence behavior of the 1-WL algorithm, each ratio convergence at some point. However, the ratios rise rapidly and dramatically for all datasets initially, indicating the strong expressiveness of the algorithm for already small numbers of iterations. Even a relatively small ratio, such as approximately 10 %, signifies a significant number of unique colors in the colorings computed by the 1-WL algorithm. It is important to consider the scale of the datasets when interpreting these ratios. For example, the MUTAG dataset consists of approximately 3 000 nodes, while the ENZYMES and IMDB-BINARY datasets contain around 20 000 nodes each. The PROTEINS dataset reaches 43 000 nodes, followed by the NCI1 dataset with 122 000 nodes, and finally, the

REDDIT-BINARY dataset with a staggering 859 000 nodes. For a comprehensive overview of the datasets and their sizes, please refer to Table 2, and for detailed information on the unique color count, also including the regression datasets and their different splits, consult Table B.7 in the Appendix.

Furthermore, it is important to note that the colors assigned by the 1-WL algorithm have no inherent structural connection between them. Even if two nodes are colored with two distinct integers that are close together, it does not imply any similarity between them. While our experiments use an implementation of the 1-WL algorithm that assigns each node the smallest unused color, a meaningful connection about the distance of colors is still absent due to shuffling the dataset and applying the 1-WL algorithm for a fixed number of iterations.

To investigate the impact of the number of 1-WL iterations on overfitting, we plotted Figure 10, which focuses solely on 1-WL+NN models and compares the different numbers of 1-WL iterations. Across all classification datasets, it is evident that the overfitting behavior becomes more substantial when increasing the number of 1-WL iterations. This finding supports our belief that the explosive increase in the number of unique colors is the primary reason for the observed overfitting behavior. In the case of ENZYMES, with an increased number of 1-WL iterations, the average training accuracy is over 60 % higher than the test accuracy, demonstrating the worsening of the overfitting behavior.



(a) ENZYMES (b) IMDB-BINARY (c) MUTAG

(d) NCI1 (e) PROTEINS (f) REDDIT-BINARY

Figure 10.: Mean difference in the classification accuracy of the training and test sets achieved by all 1-WL+NN models for each dataset. In detail, we grouped the best models according to different quantiles and the number of iterations of the 1-WL algorithm used by each model.

In conclusion, our analysis reveals that 1-WL+NN models exhibit optimal generalization when limited to one iteration of the 1-WL algorithm. Increasing the number of iterations leads to a significant overfitting behavior. This insight indicates that the expressiveness of the 1-WL

algorithm surpasses the requirements for effective learning. In comparison, GNN models excel in learning and representing graph structures in a more generalized manner. The observation that a single iteration of the 1-WL algorithm leads to the best generalization also aligns with the results from the previous section, where the models achieving the highest accuracy on the test set also utilized only a single iteration of the 1-WL algorithm (except for the NCI1 dataset). This observation raises an intriguing question: Do GNNs also attempt to compute a similarly expressive representation of all graphs, akin to a single iteration of the 1-WL algorithm? We will explore this question in the following section.

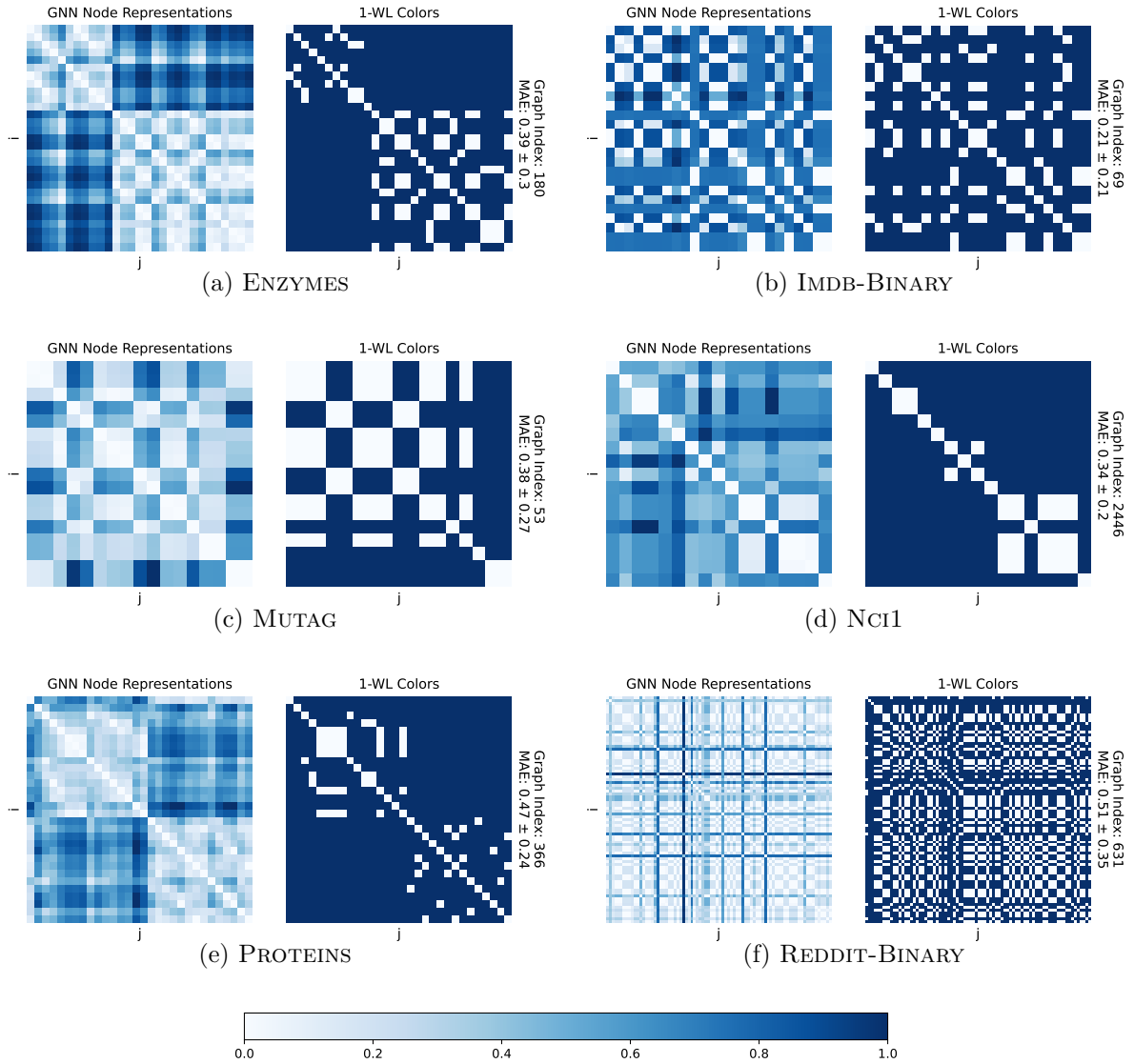
6.3. Approximating 1-WL Coloring: Investigating GNN Node Representations

This section will explore the node features computed by a GNN. Specifically, we will analyze the ability of GNNs to approximate the coloring computed by a single iteration of the 1-WL algorithm.

To visualize the approximation capabilities of the best-performing GNN models on various classification datasets, refer to Figure 11. The visualization consists of a pair of color-coded matrices for each dataset, where each pair corresponds to a single randomly drawn graph from the test set of the respective GNN model. The left matrix illustrates the distance matrix of the node representations computed by the GNN model for each pair of nodes i and j. Similarly, the right matrix represents the distances of the colorings generated by a single iteration of the 1-WL algorithm between each pair of nodes i and j. The distances are measured using the Euclidean distance and then normalized between 0 and 1 for the GNN matrix. While the distances between the 1-WL coloring are either 0 (for nodes with the same color) or 1 (for nodes with different colors) since the colorings produced by the 1-WL algorithm are discrete and do not encode any information in their distances. Further, we computed the similarity between both matrices by using their absolute difference and average these, resulting in a single value represented as the Mean Absolute Error (MAE). The MAE, along with its standard deviation and the index of the graph in the datasets, is included on the right side of the 1-WL distance matrix.

The visualization shows that the GNN representations already provide convincing approximations of the patterns observed in the 1-WL algorithm's matrix. However, it is important to note that these results are based on a single randomly selected graph from each model's test set. To obtain a more comprehensive understanding of the approximation capabilities, we repeated this process for each dataset using ten randomly selected graphs from the test set of the respective GNN model, refer to Figures B.13 to B.19 in the Appendix.

Before diving deeper into analyzing the approximation capabilities, it is crucial to highlight the significant difference between the node representations computed by a GNN and the colors computed by the 1-WL algorithm. Unlike the discrete colors with no order relation calculated by the 1-WL algorithm, the representations computed by each GNN model are multidimensional and continuous. This difference makes a perfect approximation of the 1-WL colorings challenging, as determining when two node representations are similar is not as straightforward as in the discrete 1-WL case. In the visualization, we avoided making this decision by uniformly normalizing the distances between 0 and 1 and using a continuous color bar for color coding similar nodes. However, to comprehensively evaluate each dataset entirely, we employed two approaches for tackling this problem: 1) evaluating the approximation using a threshold value for similarity (continuous approximation) and 2) translating the values into discrete distances and using the F1 score for assesing similarity (discrete approximation).



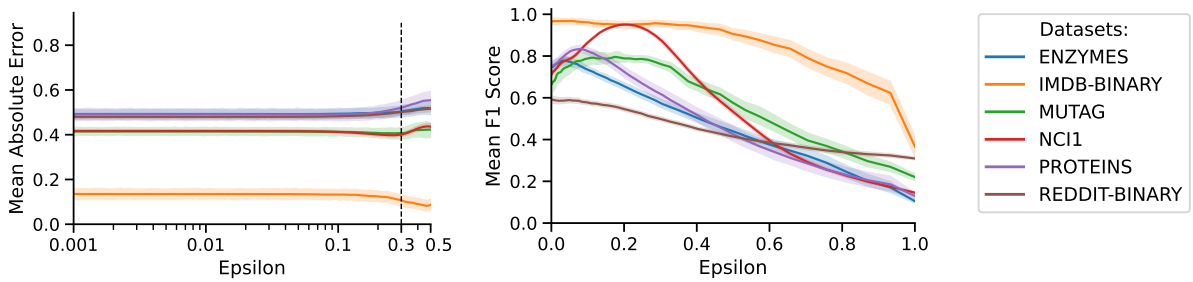
(a) Enzymes (b) Imdb-Binary

(c) Mutag (d) Nci1

(e) Proteins (f) Reddit-Binary

Figure 11.: Visualizing the performance of the best-performing GNN models of each dataset in approximating node colors computed by the 1-WL algorithm after running for one iteration. Here we randomly sampled a single graph for each dataset and visualized the approximation performance.

In the first procedure, two node representations are considered similar if their Euclidean distance is less than a threshold value, ϵ. Additionally, we consider the distances perfectly separable if their distance is greater or equal to $1 - \epsilon$. In detail, we modify the GNN distance matrix for every graph such that all distances less than ϵ are set to 0, all distances greater or equal to $1 - \epsilon$ to 1, otherwise the original distance is preserved. Afterward, we compute, as before, the MAE between each GNN matrix and the 1-WL color matrix and average all values for the entire dataset in a single MAE value in the end. We tested this procedure with various threshold values $\epsilon \in [0, 0.5)$. See Figure 12a for an illustration of the MAE for all classification datasets. Since the distances are normalized, a higher ϵ value indicates classifying



(a) Continous Approximation (b) Discrete Approximation

Figure 12.: Results obtained by using the proposed evaluation procedures to assess the ability of GNNs to approximate 1-WL colorings.

the majority of distances as negligible. As a result, the values of interest are small, which is why a logarithmic scale is used for the x-axis.

The second procedure aims to transform the regression-like values into a classification task by introducing a threshold value, ϵ, which maps distances between two node representations to a discrete value. Distances less than ϵ are classified as 0, while distances greater than or equal to ϵ are classified as 1. This process generates a distance matrix with discrete values, and the F1 score is used to evaluate the approximation capability of this newly devised matrix to the 1-WL coloring distance matrix. The process is repeated for each graph in a dataset, and the average F1 score is calculated as the final evaluation metric. Since the number of node pairs distinguished by the 1-WL algorithm differs from the number of nodes assigned the same color, an imbalance between the two labels exists. To counter any biases due to this imbalance, the F1 score is computed with macro weighting. Figure 12b provides a visualization of the results for each classification dataset.

Analyzing the figures, it becomes apparent that the approximation of the 1-WL coloring is not perfect, as indicated by the relatively high MAE and the varying F1 scores for different datasets, with the clear exception of the IMDB-BINARY dataset. Since the IMDB-BINARY and REDDIT-BINARY datasets lack node features, we initialized their features using the common procedure of encoding their node degrees. Since the coloring computed by a single iteration of the 1-WL algorithm is an encoding of the node degree, this explains the good performance of the IMDB-BINARY dataset. However, due to the large size of the REDDIT-BINARY dataset and its graphs, a one-hot encoding of the node degree is not practical. Instead, a continuous, one-dimensional node degree encoding is employed that uniformly maps a node degree into the range $[-1, 1]$. This difference in encoding is evident in the significant performance gap between these datasets in the approximation evaluation.

Interestingly, the MAE remains constant for ϵ values up to 0.3, as depicted in Figure 12a, suggesting that the GNN models tend to map nodes with different values far away from each other and vice versa for small values. This observation aligns with the low standard deviation values, indicating consistent behavior across all graphs. Moreover, to strengthen this insight, we plotted the frequency of distances that appears in the GNN distance matrices for each dataset. We calculated the frequency by creating ten uniformly, non-overlapping intervals between 0 and 1 and counted the number of distances that fall in this interval for each. Interestingly, we do not see an intense concentration of any distances in any of the intervals such that we can out rule any bias when analyzing the threshold values. Since 0.3 is already a considerable

threshold for removing distances, we infer that the GNNs compute a coloring that incorporates essential information from the 1-WL coloring while leaving out less important information. The visual representations show that the distances between nodes exhibit similar patterns as the coloring produced by the 1-WL algorithm, although not as perfect. Since the GNN models perform nearly as well as most of the 1-WL+NN models in terms of their accuracy performance, we conclude that GNNs are able to learn a more efficient encoding of the graph structure that holds the most important structural information of the 1-WL coloring.

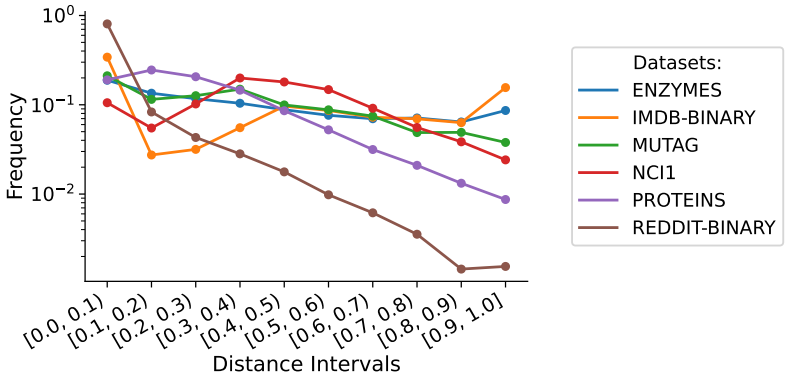


Figure 13.: Visualization of the frequency of the distance between node pairs in the representation computed by the best-performing GNN model. In detail, we divided the distance space into ten non-overlapping intervals and counted the number of occurrences that fall into this interval for each dataset.

Analyzing the other procedure, we observe that each dataset has its own optimal threshold value ϵ for the F1 score. Interestingly, these optima consistently fall within a narrow range between 0.0 and 0.2. Moreover, most datasets achieve a high F1 score, providing further evidence for our hypothesis regarding the encoding of essential structural information. With the exception of Reddit-Binary, almost all datasets achieve a score higher than 0.77. Notably, the Nci1 dataset stands out with an exceptional Mean F1 score of 0.95 for $\epsilon = 0.2$. This result indicates that the node representations computed by the best GNN model for Nci1 effectively encode the structural information captured by a single iteration of the 1-WL algorithm. Furthermore, considering the dataset taxonomy introduced in Section 5.1, Nci1 belongs to the dataset category that encodes a substantial amount of information in their graph structures, which aligns with our findings. It is also worth noting that Nci1 is the only dataset that demonstrates improved performance, in terms of accuracy, with an increased number of 1-WL iterations in the testing of the 1-WL+NN models (refer to Figure 7). Therefore we also, explored the capability of the best perfoming GNN on Nci1 to approximate the colorings computed after applying the 1-WL iteration for three iterations. The results are visualized in Figure B.16 in the Appendix. However, as explained in the previous section, the number of unique colors increases significantly with an increased number of 1-WL iterations. This has the effect that almost all nodes in a graph need to be mapped to separable node representation, which is not the case, as we see in Figure 13. This limitation applies to all datasets, leading us to conclude that GNNs cannot effectively approximate a higher number of 1-WL iterations.

In conclusion, considering the similar performance in term of accuracy of GNN and 1-WL+NN models and the relatively robust approximation capabilities of GNNs compared to the 1-WL coloring after a single iteration, we argue that GNNs can learn a more efficient encoding of

this information. Given that both frameworks achieve comparable performance on all datasets in terms of accuracy, combined with the fact that most models utilize only a single iteration of the 1-WL algorithm or an approximation, the question arises as to whether this difference remains when pooling this information using the same pooling function. This question will be investigated in the next section.

6.4. Comparing Pooled Graph Representations: GNN vs. 1-WL+NN Models

After the previous section, we observed that GNNs do not perfectly approximate the coloring calculated by the 1-WL algorithm. However, we hypothesized that GNNs might employ a more efficient encoding containing only essential information. To delve deeper into this, we compared the different graph representations derived after applying the respective pooling functions. To do this, we selected the best-performing GNN and 1-WL+NN models in terms of classification accuracy for each dataset and replaced the final MLP with various algorithms.

Table 8.: Overview of the classification accuracies achieved by the best model configuration for each dataset in percent and standard deviation. Additionally, the performance of each configuration was furhter evaluated by substituting the final MLP with either a SVM utilizing a linear kernel (SVM Linear) or the Radial Basis Function (SVM RBF), as well as the k-NN classifier with different values for k.

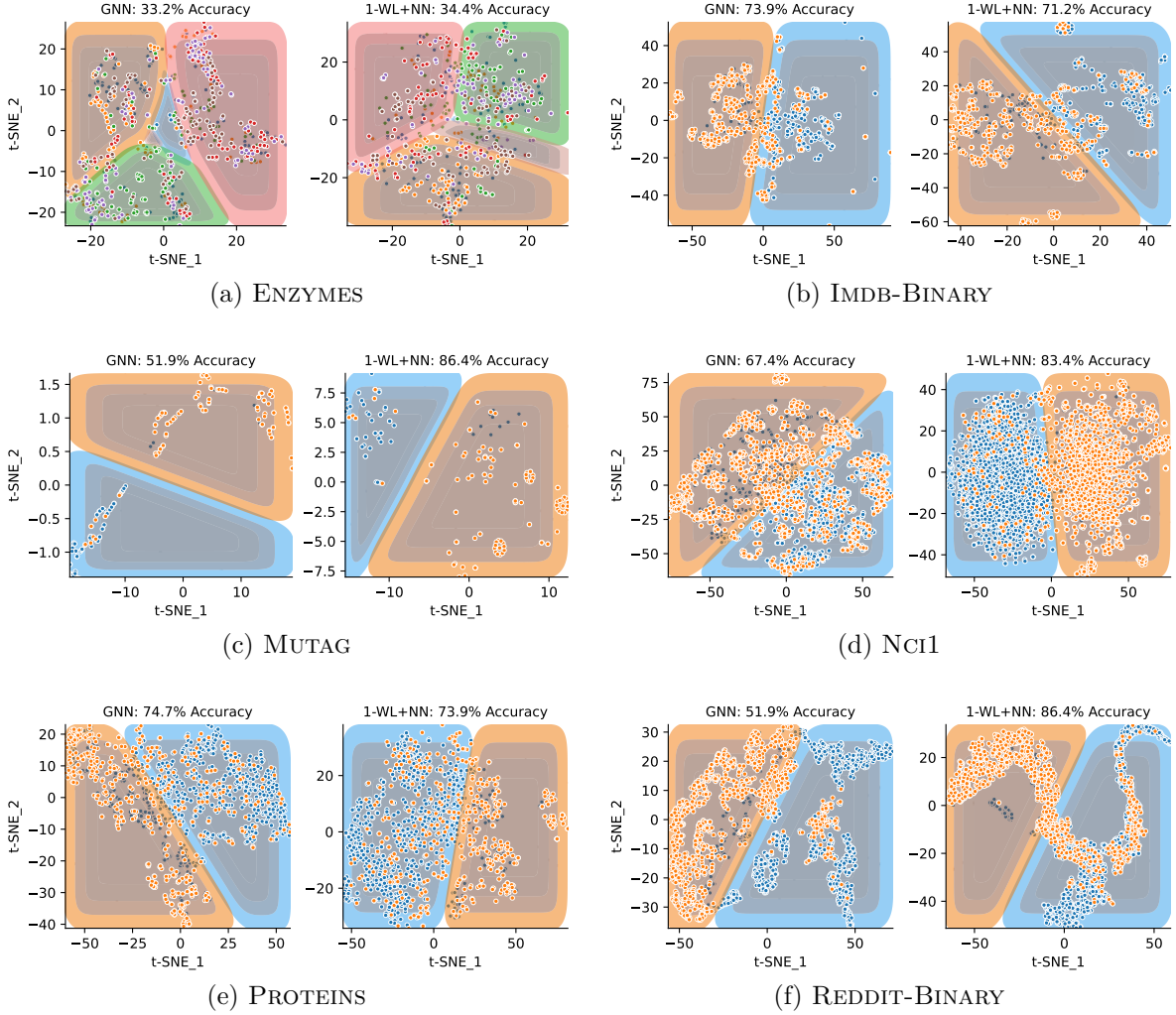
	Method	Dataset					
		Enzymes	Imdb-Binary	Mutag	Nci1	Proteins	Reddit-Binary
1-WL+NN	MLP	48.3 ±8.1	**72.4** ±4.1	85.1 ±8.6	83.6 ±4.1	**75.2** ±3.9	78.4 ±2.7
	SVM Linear	34.4 ±5.5	71.2 ±3.9	86.4 ±8.9	83.4 ±2.1	73.9 ±4.1	77.8 ±2.7
	SVM RBF	45.0 ±7.0	72.8 ±4.3	83.2 ±7.5	83.6 ±1.9	75.2 ±4.0	78.3 ±2.4
	k-NN	**56.3** ±5.8 (k=1)	72.3 ±4.1 (k=11)	**86.7** ±7.7 (k=10)	**83.9** ±1.8 (k=5)	73.9 ±4.1 (k=19)	**83.2** ±2.5 (k=13)
GNN	MLP	34.4 ±7.0	**74.7** ±3.8	84.6 ±8.7	**79.9** ±2.2	74.3 ±5.1	86.9 ±3.2
	SVM Linear	33.2 ±5.9	73.9 ±4.2	**87.4** ±6.8	67.4 ±2.2	74.7 ±4.2	84.2 ±2.8
	SVM RBF	35.9 ±6.0	74.1 ±3.9	86.0 ±7.4	73.0 ±1.9	74.6 ±4.6	88.7 ±2.4
	k-NN	**51.6** ±7.0 (k=1)	74.3 ±4.0 (k=132)	88.3 ±6.5 (k=38)	77.5 ±1.7 (k=2)	**74.9** ±4.3 (k=27)	**90.8** ±1.8 (k=15)

Specifically, we used two different configurations of Support Vector Machines (SVM): one with a linear kernel (SVM Linear) and another with a radial basis function (SVM RBF). Additionally, we employed the k-Nearest-Neighbor (k-NN) classifier for different values of k. Each algorithm was trained exclusively on the fixed output of the pooling function. To ensure fair comparisons, we froze the weights of the GNN models and the Look-Up Table of the 1-WL+NN models during this process to ensure no optimization of underlying parameters before pooling. This approach guarantees more comparable results across algorithms. Moreover, we maintained the same training, validation, and test split used for training and evaluating the MLP. For a comprehensive evaluation, we summarized the accuracies and standard deviations achieved by each method, comparing them to the MLP's performance in Table 8. Furthermore, in the case of the k-NN algorithm, we provided the achieved accuracy for the optimal k value, including the k value in the brackets. This analysis will allow us to make statements about the linear separability of the graph representations as well as how well these cluster with respect to their class labels.

Starting with the investigation of linear separability, we used the performance of SVM Linear as our basis of evaluation since this method learns a linear high-dimensional hyperplane to separate the graph representations optimally. The results of the SVM RBF serve as a baseline

that shows how effectively the data will be linear separable after applying the RBF kernel. Therefore, if both SVM methods achieve similar performance, we can conclude that the graph representations are already relatively linearly separable, and there is no need for applying an additional kernel function.

Interestingly, for both GNN and 1-WL+NN models, the linear SVM performed comparably to the MLP, with the exceptions of ENZYMES for the 1-WL+NN model and NCI1 for the GNN model. In fact, SVM Linear even outperformed the MLP of the GNN model on MUTAG, indicating the graph representations strong linear separability. Furthermore, since the insights were similar for both GNN and 1-WL+NN models, we concluded that both methods map their graph representations in such a way as to facilitate easy linear separation. Additionally, we observed that the accuracy achieved by SVM RBF is higher than the one of SVM Linear; however, this difference is negligible and insignificant (except for the ENZYMES dataset). This insight, again, demonstrates how well the graph representations are linear separability.



(a) ENZYMES (b) IMDB-BINARY

(c) MUTAG (d) NCI1

(e) PROTEINS (f) REDDIT-BINARY

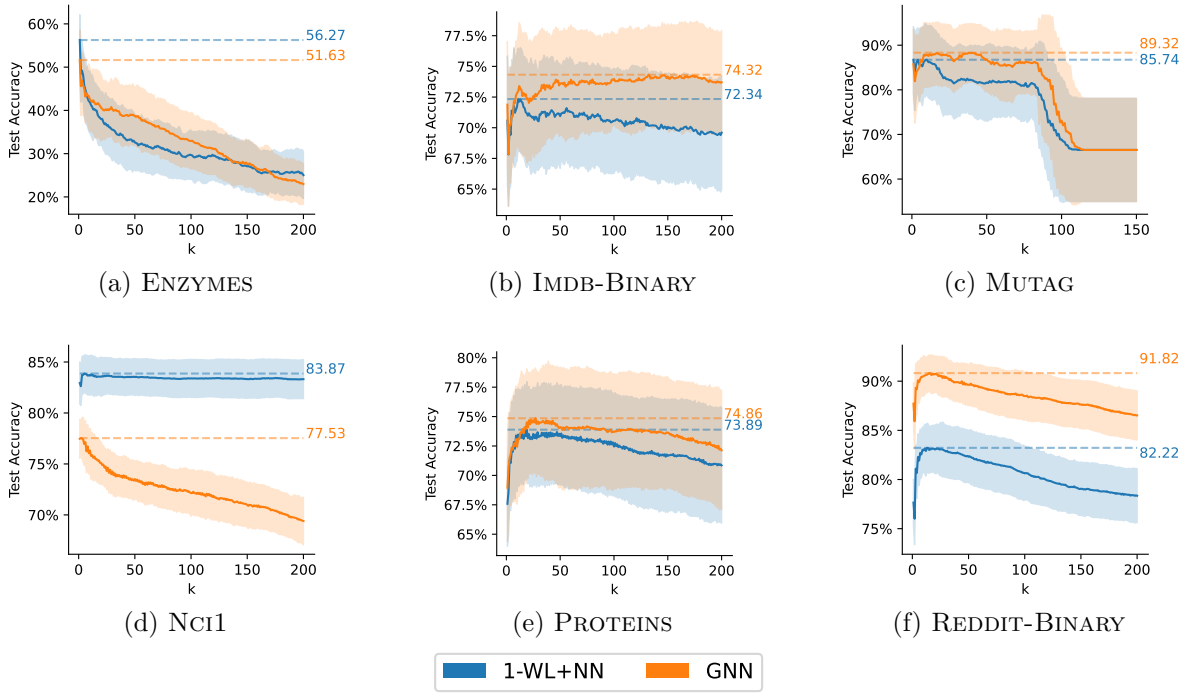
Figure 14.: Visualization of the decision boundary of each SVM Linear using t-SNE for the reduction of the dimensionality to two dimensions.

For a visualization of the decision boundary calculated by SVM Linear, see Figure 14. We

used the t-distributed stochastic neighbor embedding (t-SNE) to reduce the dimensions of the graph representation for visualization purposes. However, note that the t-SNE method distorts the actual distances between data points, such that these visualizations only aid the imagination of the actual relation between graph representations.

The insight that the GNN and the 1-WL+NN model compute node representations that, when pooled into a single graph representation, allow for good linear separability further support our hypothesis that the extra expressiveness contained in the 1-WL colorings is unnecessary, and GNNs indeed employ a more efficient encoding, as suggested in the previous section.

Another interesting property we also want to investigate is how well these graph representations cluster in their high dimensional space with respect to their class label. To do so, we utilized the k-NN classifier for different values of k with k ranging from 1 to 200 (except for the MUTAG dataset, which only contains 188 graphs, k was limited to a range of 1 to 150). Our primary interest was to observe if any range of k values exists where the accuracy consistently remained high. The existence of such a range indicates the presence of meaningful clusters among all graph representations. In order to observe such patterns, we plotted the accuracy achieved on each dataset against the corresponding k values, as illustrated in Figure 15.



(a) ENZYMES (b) IMDB-BINARY (c) MUTAG

(d) NCI1 (e) PROTEINS (f) REDDIT-BINARY

Figure 15.: 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.

Interestingly we can see for all datasets that for some range of k values, there exists a sort of plateau of the accuracy curve, except for the ENZYMES and NCI1 datasets. These plateaus corresponded to the highest accuracy achieved among all k values, indicating that both graph representations computed by 1-WL+NN and GNN models indeed form meaningful clusters with respect to their class labels. Furthermore, as the GNN model outperformed the 1-WL+NN model for most k values, we can conclude that the higher expressiveness of the node representation

comnputed by the 1-WL algorithm is obsolete and the GNN models indeed compute a more efficient encoding of its nodes, which leads to pooled graph representations that achieved the same or even better results in terms of clustering. This insight is even more pronounced in the example of the Reddit-Binary dataset, as here, a significant difference exists in its clustering behavior compared to the 1-WL+NN models.

In the case of the Enzymes dataset, allthough there is exist no such a platue, the accuracy achievable on the graph represention is for both types of models very similiar, such that the lack of indication for clustering can probaly be attributed to the dataset itself.

In the case of the Enzymes dataset, while there is no clear plateau in the curve, the accuracy achievable on the graph representations is very similar for both types of models, indicating that the lack of a distinct clustering indication may be attributed to the characteristics of the dataset.

The real exception, however, lies in the behavior of the k-NN when applied to the Nci1 dataset. Here, only the graph representations computed by the 1-WL+NN model exhibit strong clustering performance that remains consistently high across all values of k. In contrast, the graph representations of the GNN model do not seem to cluster at all, and the maximum achievable accuracy is significantly lower than the average accuracy achieved by the 1-WL+NN model representations. We attribute this effect to the 1-WL+NN model's use of three iterations of the 1-WL algorithm, compared to just one iteration used by all other models of the other datasets. Since the Nci1 dataset encodes essential information for solving its respective task structurally, this pattern might only be specific to this dataset. Moreover, we observed that no other dataset showed better performance of its 1-WL+NN models with an increased number of 1-WL iterations, as evident in Figure 7.

In conclusion, our findings indicate that the pooled graph representations computed by both 1-WL+NN and GNN models are highly linearly separable in their high-dimensional space while also forming meaningful clusters with respect to their class labels. Notably, the GNN models demonstrated superior clustering performance, providing further support for our hypothesis that the message-passing layers of a GNN encode essential information without overly emphasizing expressiveness. This suggests that the GNN models strike a balance between efficient encoding and required expressiveness depending on their application.

6.5. Preprocessing Data for GNNs

As already outlined at the beginning of this section, is that in comparison to all configurations we tested for 1-WL+NN and GNN models, the best accuracy achieved on all classification datasets is by a 1-WL+NN model, with the clear exception for the Imdb-Binary and Reddit-Binary dataset. What makes this observation so interesting is that both datasets are the only ones lacking node features, such that we initialize all GNN models working on them with an encoding of their node degrees. This difference raises the question of whether GNN models generally perform better when their input graphs undergo preprocessing. Specifically, we aimed to explore if GNN models achieve improved performance when their graphs are preprocessed using the 1-WL algorithm for a single iteration. We will refer to such GNN models as 1-WL:GNN.

This idea seems promising as most of the best-performing 1-WL+NN models also utilize a single iteration of the 1-WL algorithm as their basis for computation. Further, we know that a 1-WL:GNN model can already achieve the same accuracy as each 1-WL+NN model by simply forwarding the preprocessed input graph to the pooling function. However, we hope that 1-WL:GNN models leverage their message-passing layers to generate an improved node

representation, ultimately leading to enhanced performance in the end.

We conducted the same hyperparameter optimization routines as for the normal GNN models and listed the classification accuracy achieved by the best-performing configuration of a 1-WL:GNN model for each dataset in Table 8. We also included the best performances of each 1-WL+NN and GNN models for comparison. However, it is essential to note that due to the time constraints of this work and the prioritization of hyperparameter optimization for GNN and 1-WL+NN models, we tested fewer configurations for 1-WL:GNN models. Consequently, these results should be interpreted with caution.

Table 9.: Overview of the classification accuracies achieved by the best model configuration for each dataset in percent and standard deviation. Additionally, the performance of each configuration was furhter evaluated by substituting the final MLP with either a SVM utilizing a linear kernel (SVM Linear) or the Radial Basis Function (SVM RBF), as well as the k-NN classifier with different values for k.

Model	**Dataset**					
	Enzymes	Imdb-Binary	Mutag	Nci1	Proteins	Reddit-Binary
1-WL+NN	48.3 ±8.1	72.4 ±4.1	85.1 ±8.6	83.6 ±2.2	75.2 ±3.9	78.4 ±2.7
1-WL:GNN	35.9 ±6.9	72.0 ±4.1	83.9 ±11.0	NaN	74.5 ±4.1	84.3 ±2.8
GNN	34.4 ±7.0	74.7 ±3.8	84.6 ±8.7	79.9 ±2.2	74.3 ±5.1	86.9 ±3.2

Upon analyzing the best accuracies achieved by each 1-WL:GNN model in comparison to the other model types, we observed that its performance is not significantly better than any of the GNN or 1-WL+NN models. While it does show minor improvements compared to the GNN models, such as in the Enzymes and Proteins datasets, these improvements are not significant. However, it is important to note that the number of runs conducted for 1-WL:GNN models, 264 runs in total, is considerably smaller compared to the 5 422 runs conducted for 1-WL+NN models and 1 043 runs for GNN models. As a result, the limited number of runs might have impacted the statistical robustness of the results. For a full list of the number of runs condcuted, refer to Table B.6 in the Appendix. Further investigation with a larger number of runs could provide more conclusive insights into the performance of 1-WL:GNN models.

6.6. Impact of the Size of Datasets on the Performance of 1-WL+NN

At the beginning of our empirical investigation, an initial idea emerged when observing the first results achieved by 1-WL+NN models on the classification datasets that indicated good performance. We speculated that these results could be attributed to the small sizes of the datasets, making them particularly suitable for 1-WL+NN models. This idea also raised a question about the scalability of 1-WL+NN models on larger datasets, potentially limiting their performance compared to GNN models, which might scale more effectively. To explore this hypothesis, we sought to investigate significantly larger datasets and selected the well-known regression datasets Alchemy and Zinc. Additionally, we utilized a fixed split, significantly smaller than the original dataset, for both Alchemy and Zinc. Consequently, we tested 1-WL+NN models on both the entire dataset and the smaller split to examine potential differences in performance. An overview of the results is provided in Table 6 at the beginning of this section.

Similar to the classification datasets, we conducted a hyperparameter optimization for both 1-WL+NN and GNN models. The primary question of interest here was whether we would observe performance differences between the various splits for each dataset and if there would be discrepancies between the two types of models.

Upon analyzing these results, the first notable observation is that the best performances were achieved entirely by GNN models, specifically the GIN:Sum model. However, in comparison to the GINE-ε model by Morris et al. [2020], which we included as a reference to showcase the performance achieved by a different GNN model than ours, both our GNN and 1-WL+NN models demonstrated inferior performance. We conjecture that this discrepancy can be attributed to GINE-ε utilizing the edge features of its input graph in its computation, while our models do not.

Regarding the differences in performance between the various splits for each dataset, the evidence is not straightforward. While 1-WL+NN models performed better on the smaller split Alchemy(10k) than the full split, GNN models demonstrated a similar margin of improvement. Moreover, we observed the opposite phenomenon, with 1-WL+NN and GNN models achieving better performance on the larger Zinc split than the smaller Zinc(10k) split.

One reason why such differences were not evident for the 1-WL+NN models may be because, although the number of graphs in each dataset is significantly larger than in the classification datasets, the best performing configuration still utilized only a single iteration of the 1-WL algorithm for both Alchemy splits, two for Zinc(10k), and three for Zinc. Consequently, the number of unique colors was limited. See Table B.7 for an overview of the number of unique colors for various numbers of 1-WL iterations. In the case of the models for Alchemy, the number of unique colors remained constant at 70 for both splits, indicating that the larger number of graphs between the splits does not have an impact when employing only a single iteration of the 1-WL algorithm. However, the number of unique colors varied more dramatically for the models on the Zinc datasets, with the best model for Zinc(10k) dealing with 9 818 unique colors and the best model on Zinc encountering 290 473 unique colors.

This discrepancy can also be attributed to the splits used for Zinc and Zinc(10k), as opposed to the alchemy split, where the number of unique colors after a single iteration of the 1-WL algorithm is not equal. Specifically, for Zinc, there are 773 unique colors, and for Zinc(10k), there are only 392 unique colors. Further, it is important to note that due to time constraints and the immense time required, especially for training models on the entire Alchemy or Zinc dataset, the number of configurations we tested is relatively low compared to Zinc(10k) and Alchemy(10k). For a complete overview of the number of runs for each dataset and each model, refer to Table B.6 in the Appendix.

In conclusion, the impact of the size of a dataset on the performance of 1-WL+NN models must be investigated further. In detail, future research needs to investigate datasets not only of large size but also datasets that consist of large graphs, such that the number of unique colors is already relatively high even after a single iteration of the 1-WL algorithm. A good example of such a dataset is the Reddit-Binary dataset, where the number of unique colors after a single iteration of the 1-WL algorithm is 566 due to the high number of nodes on average per graph. For this dataset, a performance gap between 1-WL+NN and GNN models can be observed, with the best GNN model achieving up to an 8 % higher accuracy than the best 1-WL+NN model. However, it is also essential to acknowledge that the reasons for this performance gap are not solely attributable to the dataset's size, as other factors are at play, as discussed in the preceding subsections.

7. Discusssion

Introdcution for this section!

7.1. Learned Lessons

- 1-WL+NN is a very good tool for investigating GNNs
  - many imperical simalarities next to the theoretical equivalence:
    - Similiar accuracy performance
    - approximation of coloring / node representations
    - linear separability and clustering
  - However point out that 1-WL+NN is simply a lab tool -> not suited for real world applications due to a absence of a global 1-WL alogirhtm -> GNNs are in this regard way more flexible
  - Emphasis that our implementation of the 1-WL+NN algorithm however learn way faster than the GNN models as only the MLP has to be optimized -> therefore good model for retrieving first information about a new and unknown dataset.
- Summarize all GNN insights:
  - Allthough GNNs are theoretically as expressive as the 1-WL alogrithm, the node features and consequently its graph representations only make use of this expressiveness if needed. For example, the NCI1 datasets that encoded most of its crucial information structurally (taxonomy and findings of minimal wl iterations for achieving optimal accuracy), had the best score in the discrete approximation metric of the 1-WL colorings -> hence GNNs make use of its expressiveness but only if needed.
  - Small overfitting behaviour in comparison to 1-WL+NN models -> good generalization
  - robust representations ->
  - Preprocessing does not seem to really help to increase the accuracy -> achieve same accuracy with no significant improvements
  - graph representations are good linear seperable and cluster very well -> might be woth checking out other postprocessing steps than a standard MLP

7.2. Future Work

- test other GNN architectures
- investigate impact of datasets size -> In particular check if this is only an effect of regression or does this also apply to classificaiton -> candidates are for exmaple MalNet adn MalNet tiny
- Noising von datasets schauen wie 1-WL+NN and GNN damit copen
- More investigation in 1-WL:GNN-> more runs
- Investigate 1-WL for continous cases such that it applies better for the regression datasets

7.3. Conclusion

Conclude the Thesis!

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 *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

[5] L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *20th annual symposium on foundations of computer science (sfcs 1979)*, pages 39–46. IEEE, 1979.

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] 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.

[8] 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.

[9] X. Bresson and T. Laurent. A two-step graph convolutional decoder for molecule generation. *arXiv preprint arXiv:1906.03412*, 2019.

[10] 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.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[12] 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.

[13] A. Cardon and M. Crochemore. Partitioning a graph in o (¦ a¦ log2¦ v¦). *Theoretical Computer Science*, 19(1):85–98, 1982.

[14] 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.

[15] Z. Chen, S. Villar, L. Chen, and J. Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32, 2019.

[16] 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.

[17] 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.

[18] 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.

[19] F. Geerts. The expressive power of kth-order invariant graph networks, 2020.

[20] 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*, pages 1263–1272. PMLR, 2017.

[21] M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

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

[23] 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.

[24] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[25] N. Immerman and E. Lander. *Describing Graphs: A First-Order Approach to Graph Canonization*, pages 59–81. Springer, 1990.

[26] 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.

[27] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[28] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[29] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[30] 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.

[31] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[32] 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.

[33] 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 *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[34] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL `www.graphlearning.io`.

[35] 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.

[36] G. Nikolentzos, M. Chatzianastasis, and M. Vazirgiannis. What do gnns actually learn? towards understanding their representations. *arXiv preprint arXiv:2304.10851*, 2023a.

[37] 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.

[38] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on computing*, 16(6):973–989, 1987.

[39] 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.

[40] 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.

[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, 2008.

[42] 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.

[43] 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.

[44] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[46] 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.

[47] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[48] 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.

[49] 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.

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

[51] 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.

[52] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[53] 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.

[54] 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. Preliminaries

1.1. Graph Attention Network

The Graph Attention Network(GAT) developed by [45] is unique and stands out from other GNN architectures due to its unique attention mechanism. The attention mechanism is inspired by the concept of attention found in the field of natural language processing, specifically in the work [6], which allows a model to focus on specific parts of the input data that are more relevant for the given task. In the case of GAT, this attention mechanism is adapted to graphs, enabling the model to learn the importance of each neighboring node during the update step of each node in a message-passing layer.

Definition 26. The message-passing layers of the GAT architecture are defined as follows:

$$f_{\text{merge}}^{(t)} = \sigma(\alpha_{vv} \cdot f^{(t)}(v) + f_{\text{agg}}^{(t)}), \quad \text{and} \quad f_{\text{agg}}^{(t)} = \sum_{u \in \mathcal{N}(v)} \alpha_{vu} \cdot W^{(t)} \cdot f^{(t-1)}(u),$$

with the attention coefficient α_{vv} computed as follows:

$$\alpha_{vu} = \frac{\exp\Big(\text{LeakyReLU}\Big(\vec{a}^T \cdot \mathsf{concat}[W^{(t)} f^{(t-1)}(v),\ W^{(t)} f^{(t-1)}(u)]\Big)\Big)}{\sum_{k \in \mathcal{N}(v) \cup \{v\}} \exp\big(\text{LeakyReLU}\big(\vec{a}^T \cdot \mathsf{concat}[W^{(t)} f^{(t-1)}(v),\ W^{(t)} f^{(t-1)}(k)]\big)\big)},$$

where $\vec{a}$ is a learnable vector, $W^{(t)}$ a learnable matrix, and σ a non-linear activation function. Further, the LeakyReLU function is defined as follows:

$$\text{LeakyReLU}(x) := \begin{cases} x, & \text{if } x \geq 0 \\ m \cdot x, & \text{else} \end{cases},$$

where m is a learnable parameter and is referred to as "negative-slop". This value is in the context of the GAT usually initialized to $m := 0.2$.

2. Theoretical Connection

2.1. Lemma 19: Composition Lemma for 1-WL+NN

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

Definition 27 (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 MLP is defined as $\mathsf{MLP} := (\mathsf{MLP})_{t \in [T]}$ where T is the number

of layers. For every $t \in [T]$, the t.th layer of the MLP is the t.th item in the finite sequence $(\mathsf{MLP})_t$. Further, all layers are recursively defined on any input v as:

$$\begin{aligned}(\mathsf{MLP})_0(v) &\coloneqq v \\ (\mathsf{MLP})_{t+1}(v) &\coloneqq \sigma_t(W_t \cdot (\mathsf{MLP})_t(v) + b_t), \quad \forall t \in [T-1]\end{aligned}$$

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

Having established a formal definition and notation, we will now prove the 1-WL+NN composition lemma.

Proof of Lemma 19. Let $\mathcal{C}$ be a collection of functions computed by 1-WL+NN, $h_1, \ldots, h_n \in \mathcal{C}$, and $\mathsf{MLP}^\bullet$ a multilayer perceptron. Further, let $f_1, \ldots, f_n$ be the encoding functions, as well as $\mathrm{MLP}_1, \ldots, \mathrm{MLP}_n$ be the multilayer perceptrons used by $h_1, \ldots h_n$ respectively. As outlined in Figure 5, we will now construct f^* and MLP^*, such that for all graphs $G \in \mathcal{X}$:

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

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

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

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

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

Using the decomposition introduced in Definition 27, we can decompose each MLP_i for $i \in [n]$ at layer $j > 0$ as follows: $(\mathsf{MLP}_i)_j(v) := \sigma_{i,j}(W_j^i \cdot (\mathsf{MLP}_i)_{j-1}(v) + b_j^i)$. Using this notation we construct MLP^* as follows:

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

where T is the maximum number of layers of the set of MLP_i's, and $T^\bullet$ is the number of layers of the given $\mathsf{MLP}^\bullet$. Thereby, we define in the first equation the start of the sequence as the input; with the second line, we construct the "simultaneous" execution of the MLP_i's, and in the last equation line, we add the layers of the given $\mathsf{MLP}^\bullet$ 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 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 MLP_i. And finally, we define the overall activation function σ_j^* as following:

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

where d_i is the dimension of the output of 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 MLP_i's individually to their respective computation. □

B. Appendix Part II

1. Testing Configuration

1.1. Definition of the Normalized Shannon-Index

We used the following definition of the Normalized Shannon-Index.

Definition 28 (Normalized Shannon-Index)**.** The metric is computed as follows:

$$-\frac{1}{\log_2(|C|)} \cdot \sum_{i \in C} \frac{n_i}{n} \cdot \log_2(\frac{n_i}{n}) \tag{1.1}$$

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

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	-	-	-
Social networks	COLLAB	0.61	0.98	-	-	-	-
	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

Overview of Hyperparameters: 1-WL+NN on the Classification Datasets

Table B.2.: Listing of the hyperparameter with which we configured the 1-WL+NN models for each classification dataset. In terms of accuracy, the choice of hyperparameters that led to the best-performing configuration is highlighted in boldface if there are numerous options for a hyperparameter.

Hyperparameter	**Dataset**					
	ENZYMES	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Batch Size	32	32	32	33	32	32
Learning Rate	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$
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	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$
Embedding Dimension	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$
Pooling function	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}

Overview of Hyperparameters: GNN on the Classification Datasets

Table B.3.: Listing of the hyperparameter with which we configured the GNN models for each classification dataset. In terms of accuracy, the choice of hyperparameters that led to the best-performing configuration is highlighted in boldface if there are numerous options for a hyperparameter.

Hyperparameter	**Dataset**					
	ENZYMES	IMDB-BINARY	MUTAG	NCI1	PROTEINS	REDDIT-BINARY
Batch Size	32	32	32	$\{33, 129\}$	32	32
Learning Rate	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$	$X \sim U(0.0001, 0.1)$
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}	{GAT, GCN, GIN}	{GAT, GCN, GIN}	{GAT, GCN, GIN}	{GAT, GCN, GIN}	{GAT, GCN, GIN}
GNN Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
GNN Dropout	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$
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	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$
Pooling function	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}

Overview of Hyperparameters: 1-WL+NN on the Regression Datasets

Table B.4.: Listing of the hyperparameter with which we configured the 1-WL+NN models for each regression dataset. In terms of accuracy, the choice of hyperparameters that led to the best-performing configuration is highlighted in boldface if there are numerous options for a hyperparameter.

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	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$	$X \sim U(0, 0.2)$
Embedding Dimension	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$	$\{\text{None}, 16, 32, 64, 128\}$
Pooling function	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}

Overview of Hyperparameters: GNN on the Regression Datasets

Table B.5.: Listing of the hyperparameter with which we configured the GNN models for each classification dataset. In terms of accuracy, the choice of hyperparameters that led to the best-performing configuration is highlighted in boldface if there are numerous options for a hyperparameter.

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
GNN Architecture	GIN	GIN	GIN	GIN
GNN Activation Function	ReLU	ReLU	ReLU	ReLU
GNN Dropout	0.0	0.0	0.0	0.0
GNN Hidden Dimension	256	256	256	256
GNN Jumping-Knowledge	cat	cat	cat	cat
GNN Number of Layers	5	5	5	5
MLP Activation Function	ReLU	ReLU	ReLU	ReLU
MLP Normalization	BatchNorm	BatchNorm	BatchNorm	BatchNorm
MLP Number of Layers	4	4	4	4
MLP Dropout	0.0	0.0	0.0	0.0
Pooling function	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}	{Max, Mean, Sum}

1.4. Performance overview for each Hyperparameter

In this section, we present the results of our hyperparameter optimization for the 1-WL+NN framework on each classification dataset. We plot a random subset of the tested configurations where each line in the visualization represents a single configuration and is color-coded based

on its accuracy relative to the other plotted configurations. Bright yellow lines indicate configurations that performed among the best, while dark purple lines represent configurations with the worst performance. The color coding gradually transitions between these two color endpoints, allowing for a clear visual representation of the performance spectrum.

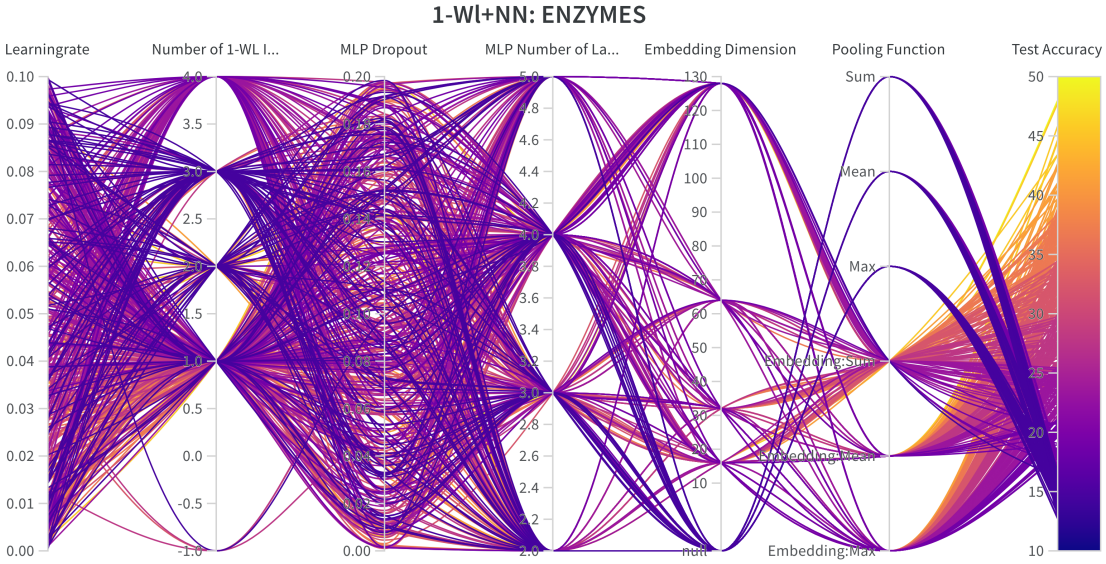


Figure B.1.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured 1-WL+NN model for the ENZYMES dataset.

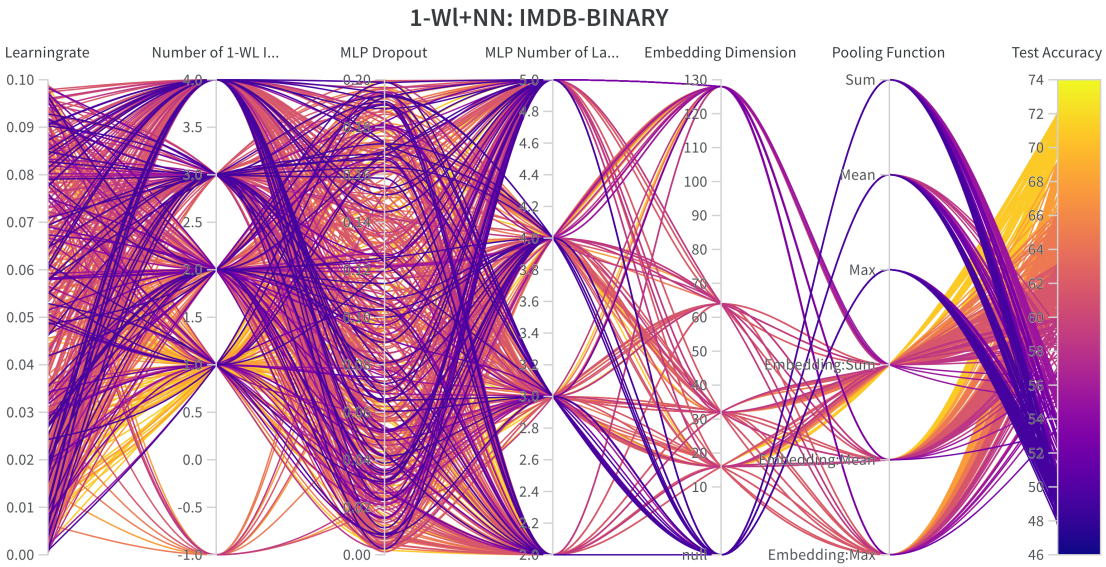


Figure B.2.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured 1-WL+NN model for the IMDB-BINARY dataset.

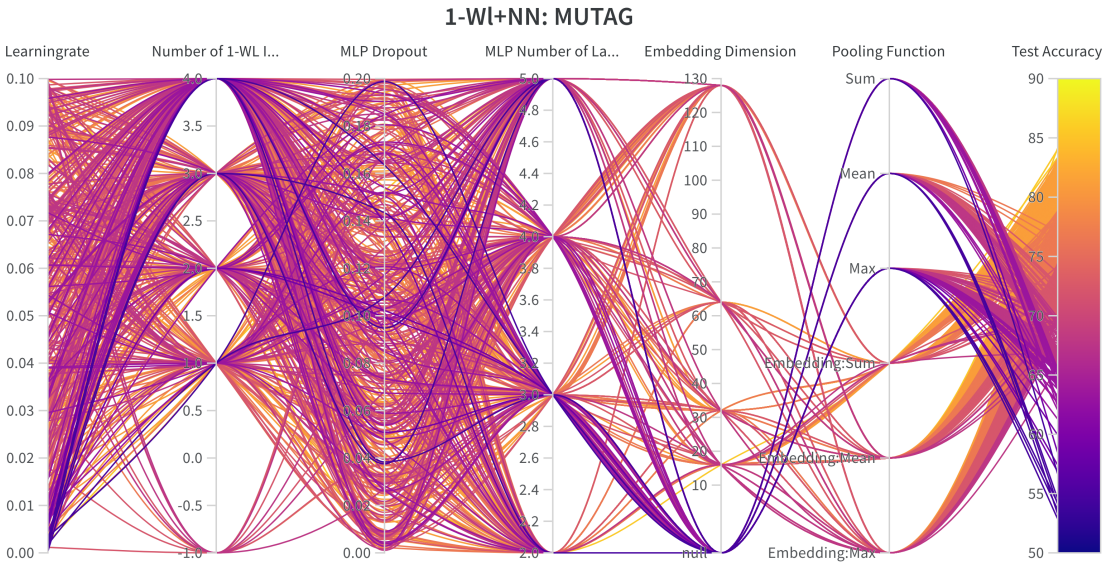


Figure B.3.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured 1-WL+NN model for the MUTAG dataset.

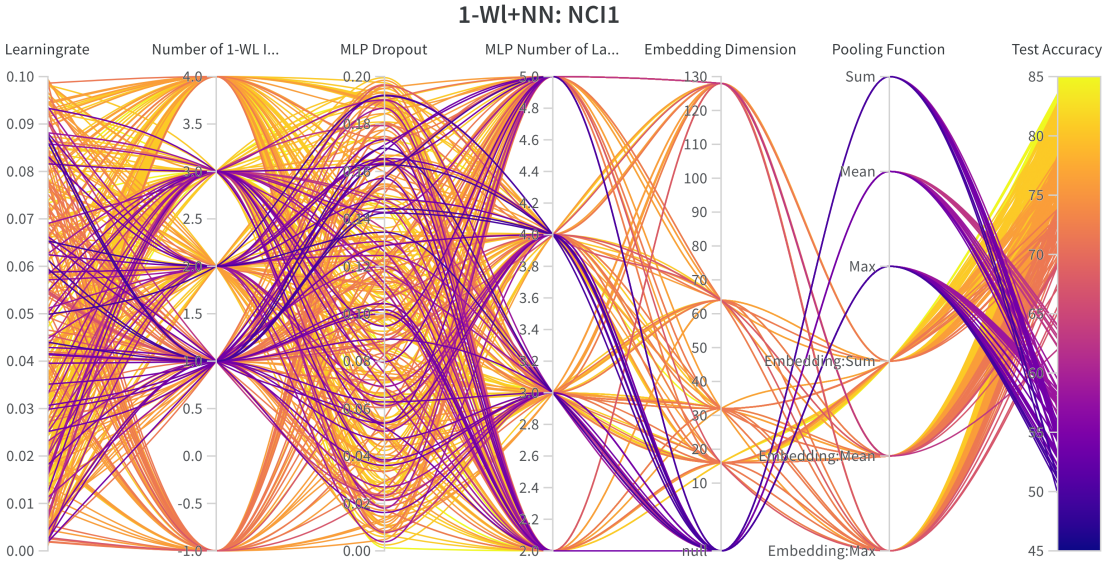


Figure B.4.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured 1-WL+NN model for the NCI1 dataset.

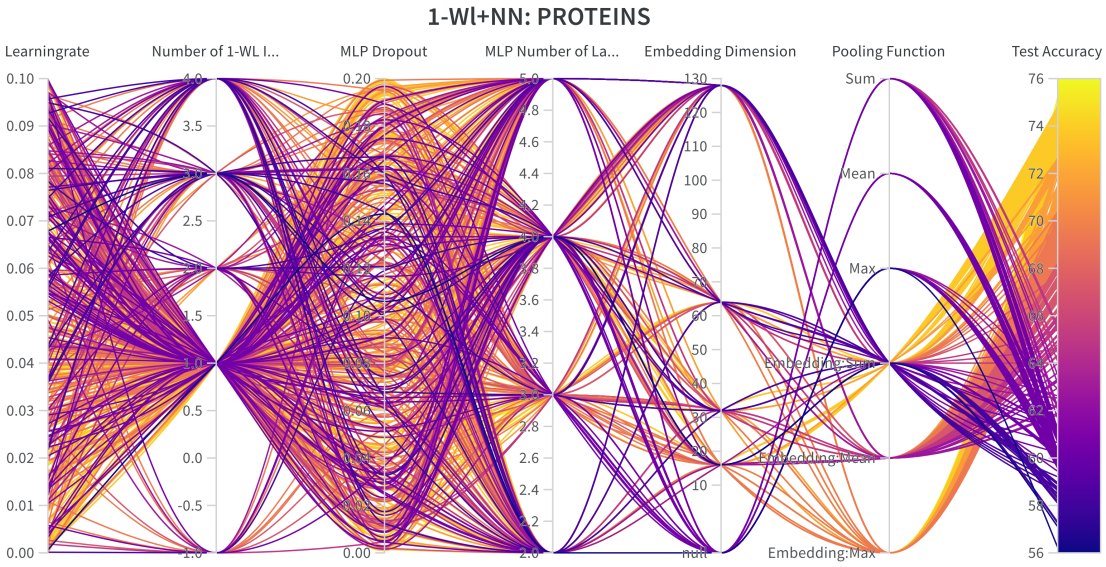


Figure B.5.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured 1-WL+NN model for the PROTEINS dataset.

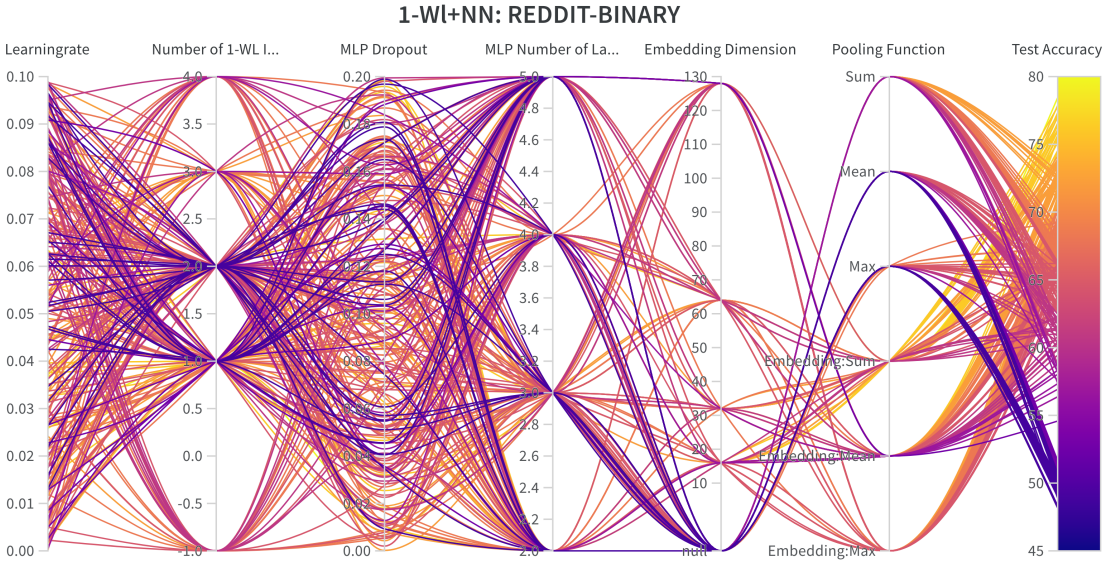


Figure B.6.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured 1-WL+NN model for the REDDIT-BINARY dataset.

1.5. Overview of the Impact of each Hyperparameter for GNN Models

In this section, we present, similarly to the previous section, the results of our hyperparameter optimization for the GNN framework on each classification dataset. We plot a random subset of the tested configurations where each line in the visualization represents a single configuration and is color-coded based on its accuracy relative to the other plotted configurations. Bright yellow lines indicate configurations that performed among the best, while dark purple lines

represent configurations with the worst performance. The color coding gradually transitions between these two color endpoints, allowing for a clear visual representation of the performance spectrum.

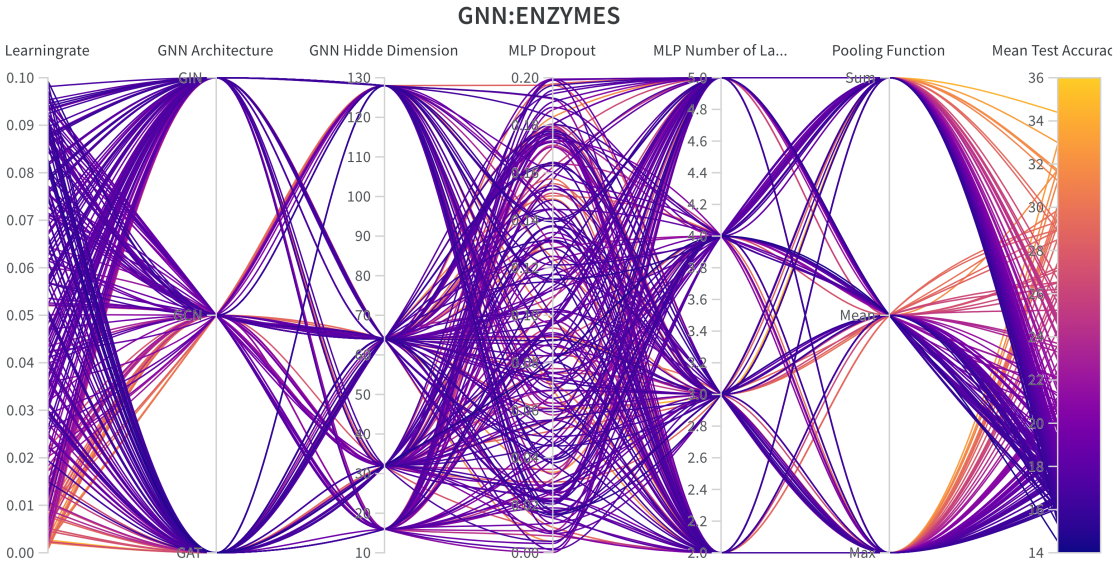


Figure B.7.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured GNN model for the ENZYMES dataset.

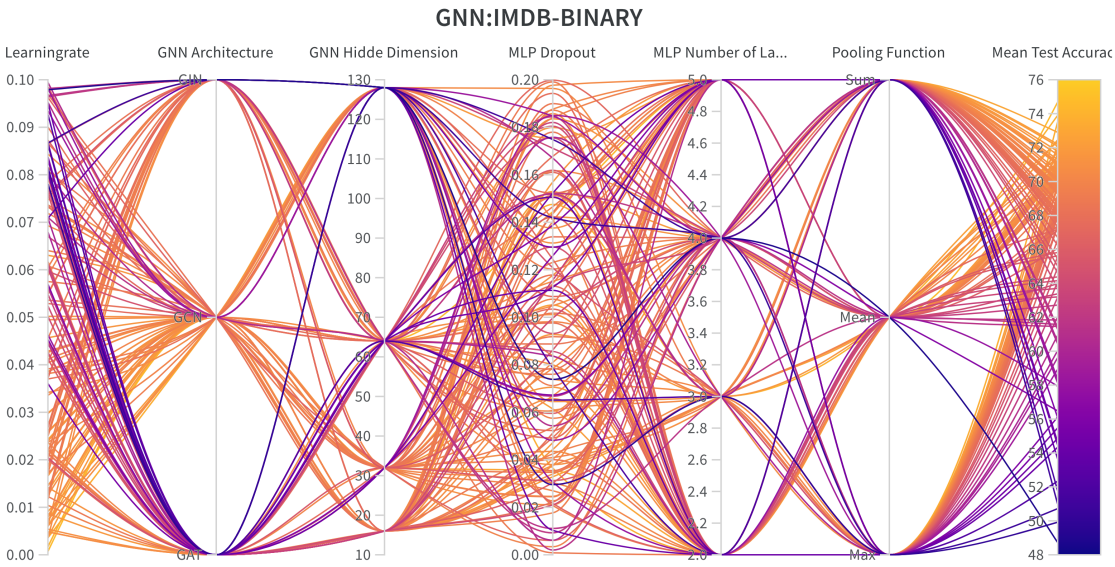


Figure B.8.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured GNN model for the IMDB-BINARY dataset.

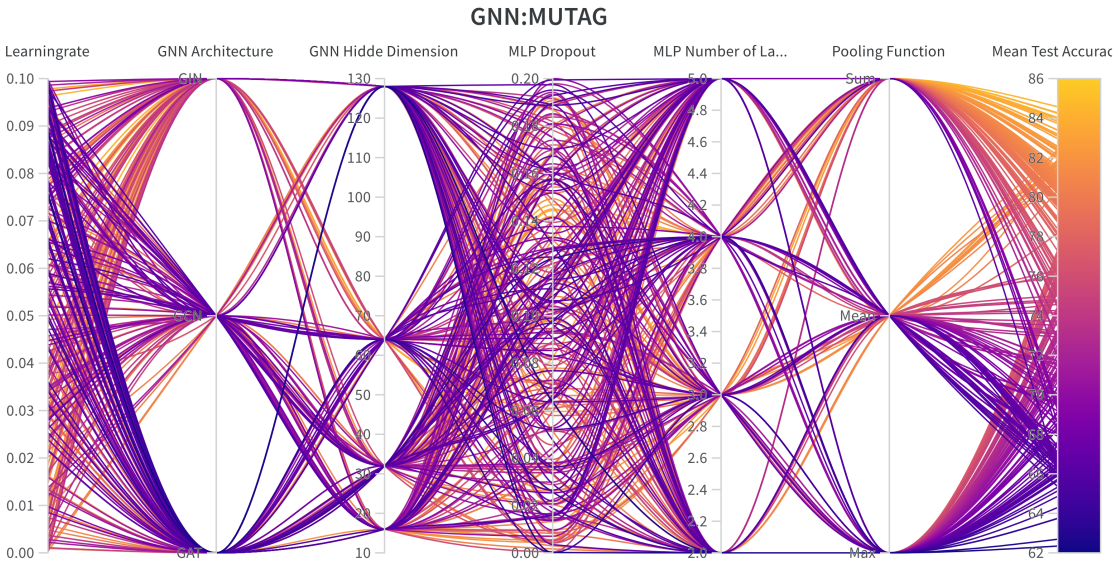


Figure B.9.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured GNN model for the MUTAG dataset.

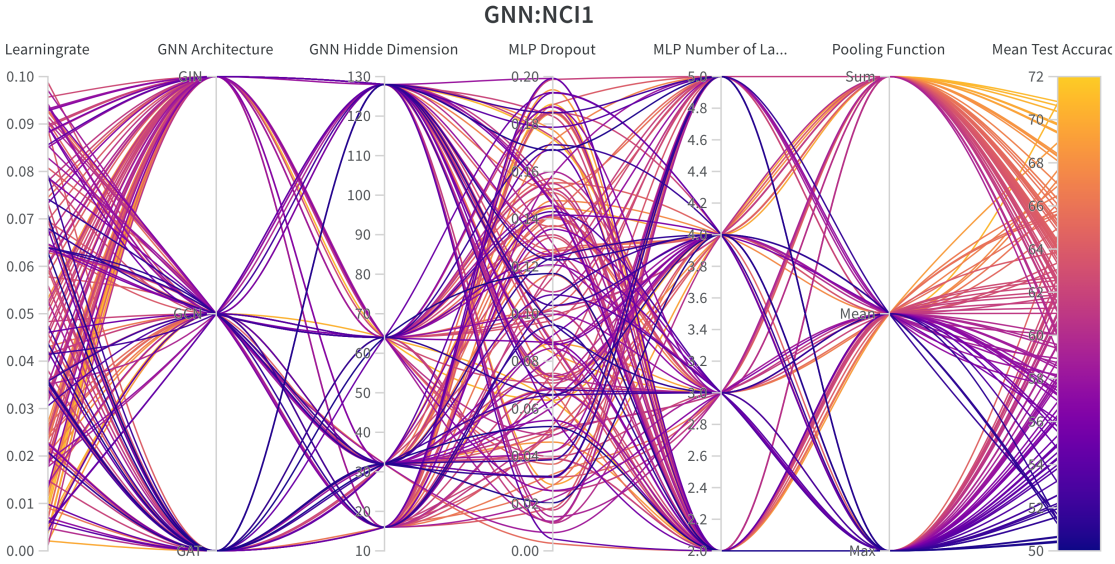


Figure B.10.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured GNN model for the NCI1 dataset.

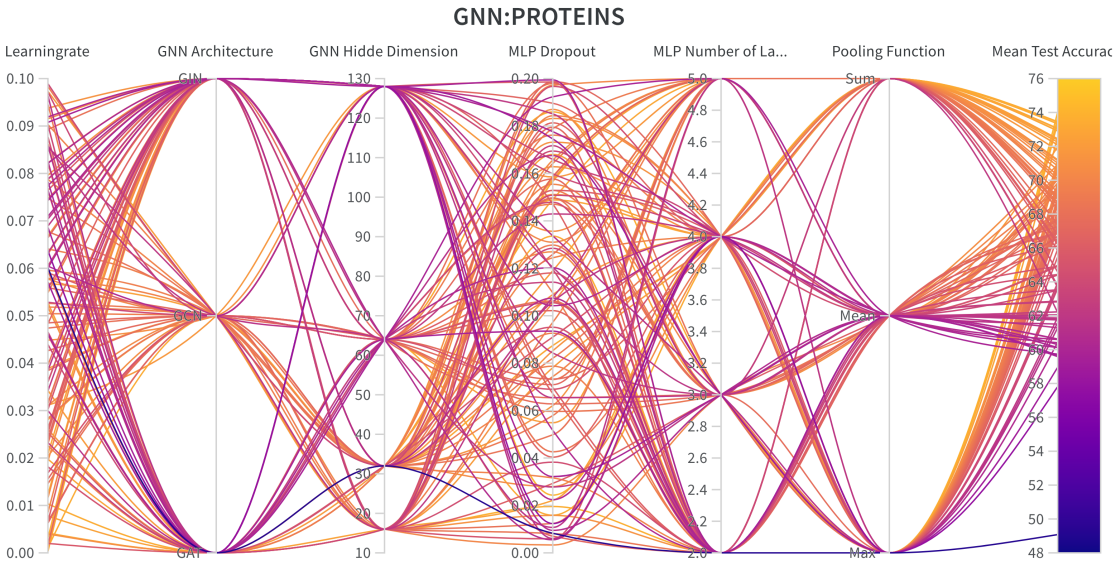


Figure B.11.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured GNN model for the PROTEINS dataset.

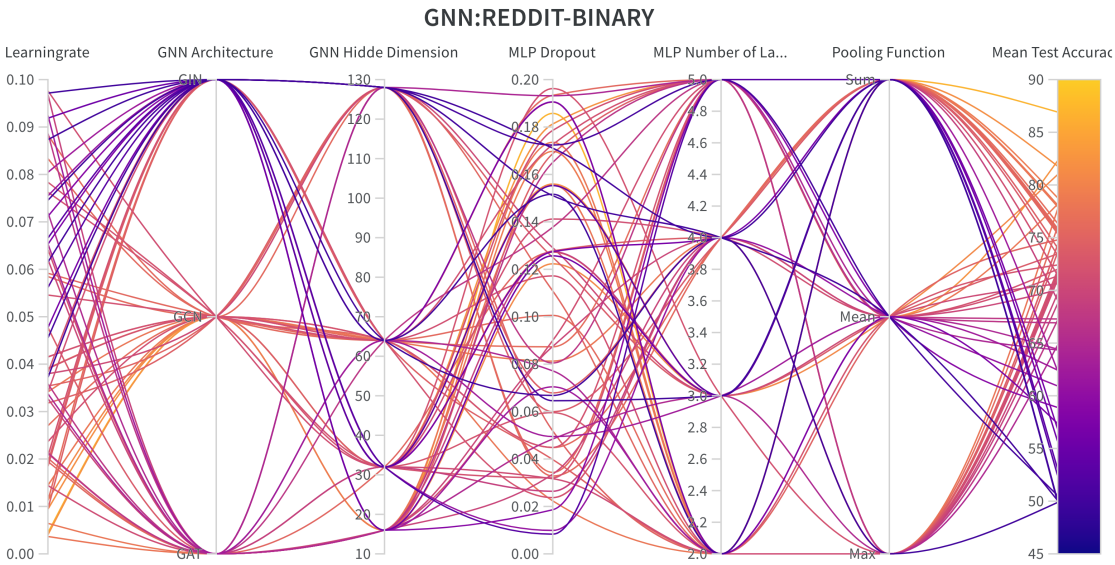


Figure B.12.: Overview of the effects of each hyperparameter on the accuracy of the corresponding configured GNN model for the REDDIT-BINARY dataset.

1.6. Overview of the Number of Configurations

Table B.6.: Overview of the number of different configurations tested for each model type and each dataset.

	Method	Dataset									
		Classification						Regression			
		Enzymes	Imdb-Binary	Mutag	Nci1	Proteins	Reddit-Binary	Alchemy	Alchemy(10k)	Zinc	Zinc(10k)
1-WL+NN	Max	86	70	150	26	35	40	0	0	0	0
	Mean	76	67	120	19	27	40	0	0	0	0
	Sum	85	67	130	14	29	41	0	0	0	0
	Embedding-Max	338	282	290	79	245	45	3	95	6	33
	Embedding-Mean	288	271	299	109	216	50	6	77	8	24
	Embedding-Sum	296	293	302	79	215	48	5	273	7	18
Graph Neural Networks	GAT:Max	17	17	36	11	10	6	0	0	0	0
	GAT:Mean	28	15	29	12	10	6	0	0	0	0
	GAT:Sum	26	17	37	15	16	6	0	0	0	0
	GCN:Max	31	22	37	17	10	9	0	0	0	0
	GCN:Mean	19	26	26	19	16	5	0	0	0	0
	GCN:Sum	30	29	27	17	14	10	0	0	0	0
	GIN:Max	21	17	28	25	20	3	1	1	1	1
	GIN:Mean	31	9	31	18	26	9	2	2	2	2
	GIN:Sum	26	9	36	20	11	9	1	1	1	1
1-WL:GNN	GAT:Max	2	4	1	9	4	2	0	0	0	0
	GAT:Mean	3	4	6	6	7	4	0	0	0	0
	GAT:Sum	5	5	6	9	4	4	0	0	0	0
	GCN:Max	2	6	4	12	6	3	0	0	0	0
	GCN:Mean	3	5	3	10	8	2	0	0	0	0
	GCN:Sum	6	2	3	5	5	3	0	0	0	0
	GIN:Max	3	4	4	9	7	5	0	0	0	0
	GIN:Mean	2	6	2	10	7	3	0	0	0	0
	GIN:Sum	4	2	4	11	4	4	0	0	0	0

2. Empirical Testing

2.1. Approximating 1-WL Coloring: Investigating GNN Node Representations

Table B.7.: Overview of the number of unique colors in the colorings computed by the 1-WL algorithm when applied to each dataset. Specifically, we specified the number of iterations of the 1-WL algorithm. Additionally, the "# Nodes" column showcases the maximum number of unique colors that can be appear in the colorings.

	Dataset	Number of 1-WL Iterations											
		0	1	2	3	4	5	6	7	8	9	10	# Nodes
Classification	Enzymes	2	231	10 416	15 208	16 029	16 450	16 722	16 895	17 026	17 130	17 204	19580
	Imdb-Binary	1	65	2 931	3 595	3 595	3 595	3 595	3 595	3 595	3 595	3 595	19 773
	Mutag	2	33	174	572	1 197	1 766	2 167	2 403	2 511	2 560	2 579	3 371
	Nci1	2	292	4 058	22 948	44 508	58 948	68 632	75 754	81 263	85 590	88 968	122 747
	Proteins	2	297	20 962	35 676	37 940	38 653	38 926	39 064	39 141	39 180	39 203	43 471
	Reddit-Binary	1	566	71 893	244 529	317 728	333 258	335 961	336 412	336 490	336 506	336 507	859 254
Regress.	Alchemy	2	70	4 782	164 224	620 332	995 264	1 166 951	1 216 094	1 225 861	1 227 632	1 227 904	2 046 329
	Alchemy(10k)	2	70	2 764	33 903	76 822	98 394	104 687	105 907	106 109	106 149	106 166	121 422
	Zinc	2	773	288 74	290 473	1 000 917	1 977 437	2 921 087	3 690 341	4 270 959	4 681 881	4 945 363	5 775 257
	Zinc(10k)	2	392	9 818	61 198	132 862	185 699	216 210	233 484	242 866	247 688	249 971	278 179

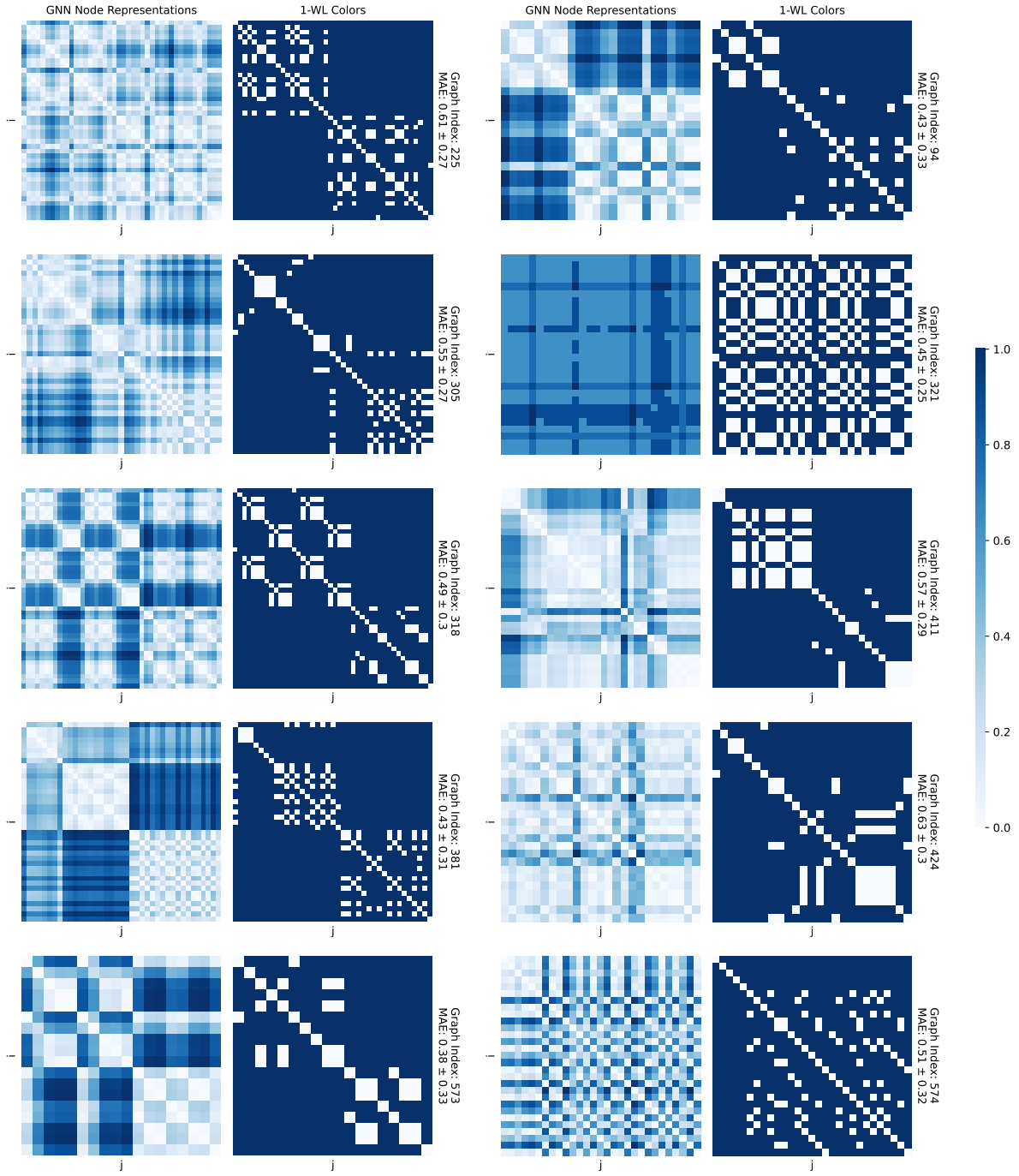


Figure B.13.: Visualizing the performance of the best performing GNN on the ENZYMES dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN's test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is 0.49 ± 0.3.

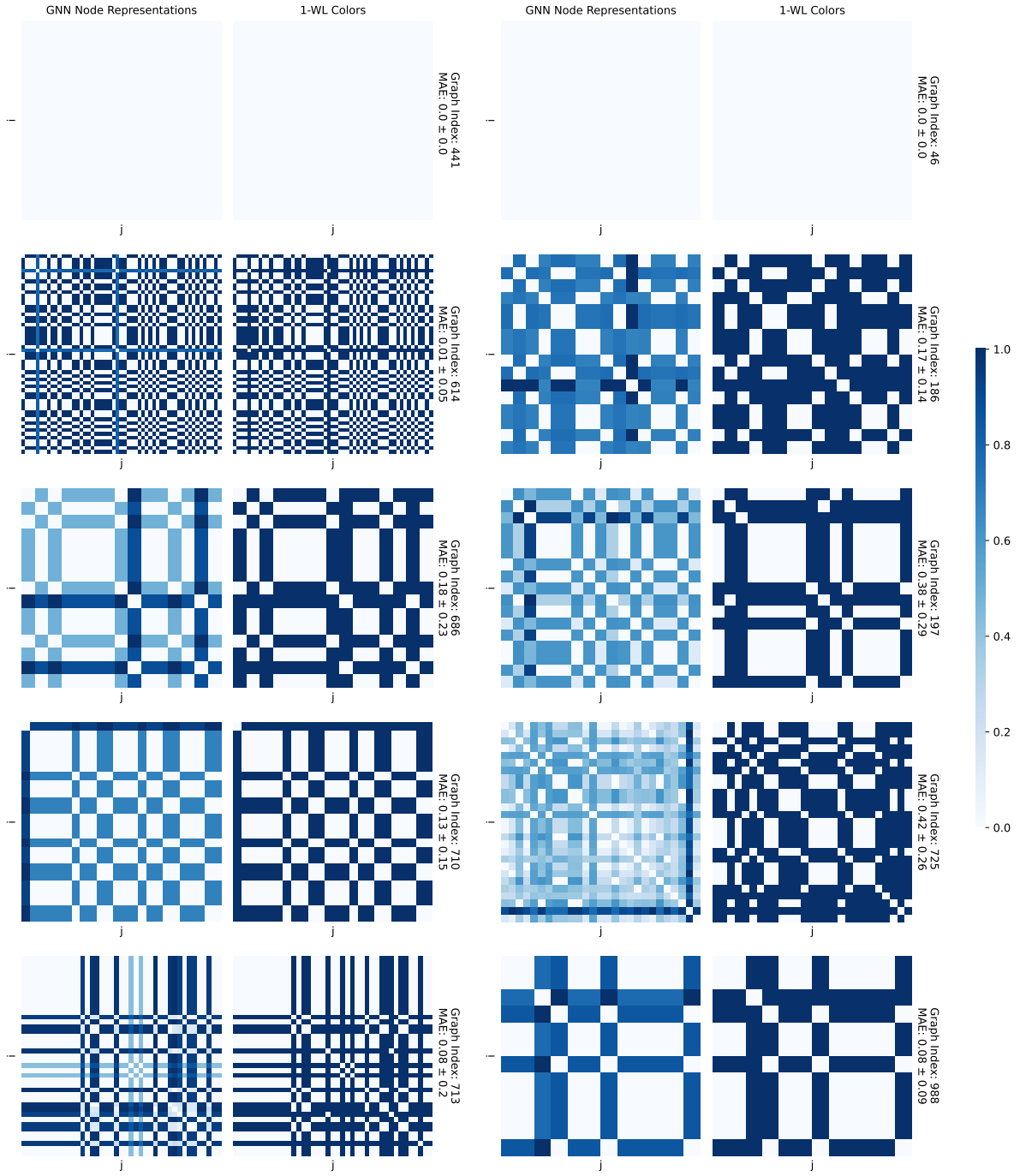


Figure B.14.: Visualizing the performance of the best performing GNN on the IMDB-BINARY dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN's test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is 0.14 ± 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.

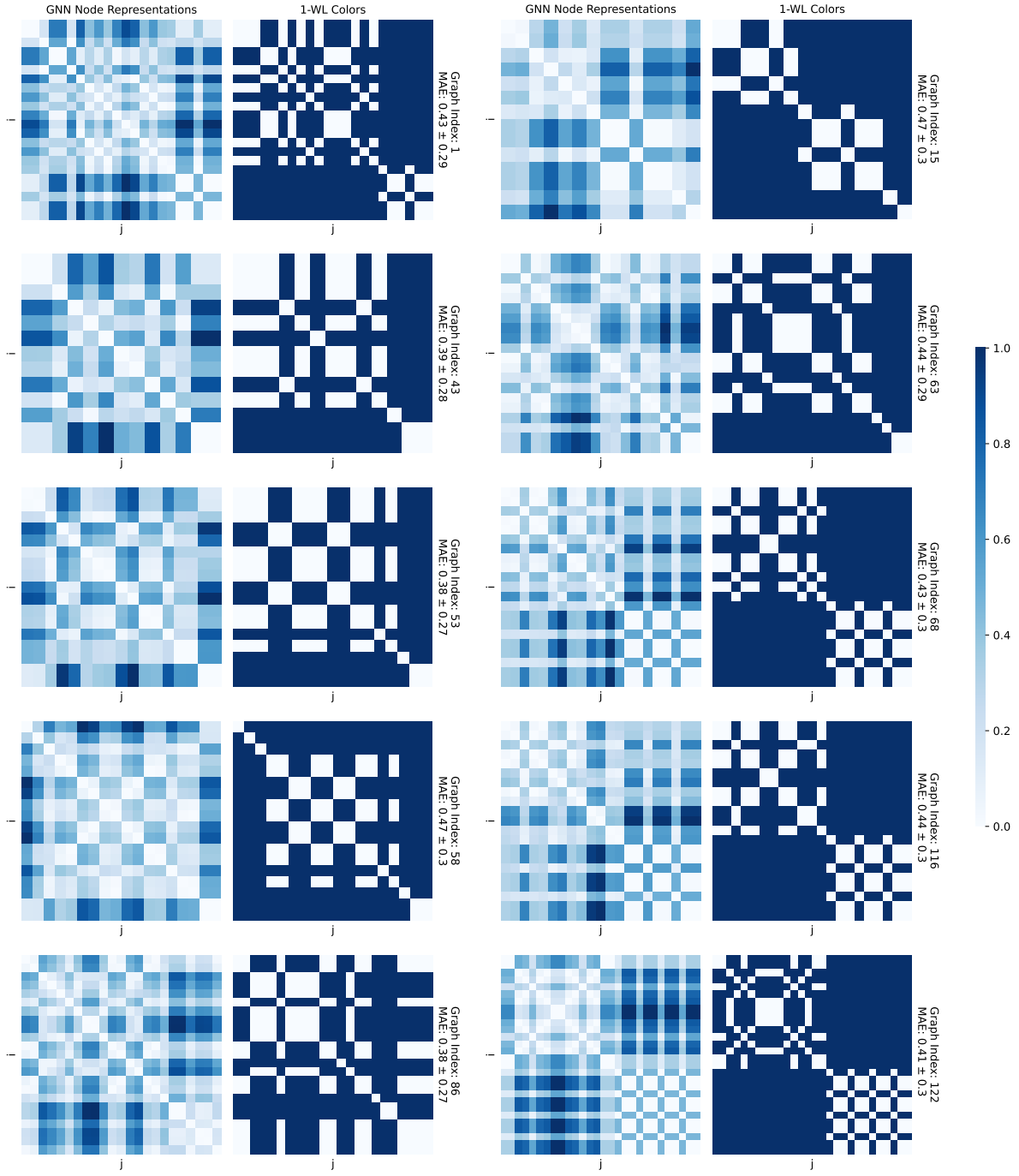


Figure B.15.: Visualizing the performance of the best performing GNN on the MUTAG dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN's test set, and the 1-WL algorithm ran only for three iteration. The average error for the entire test set is 0.42 ± 0.29.

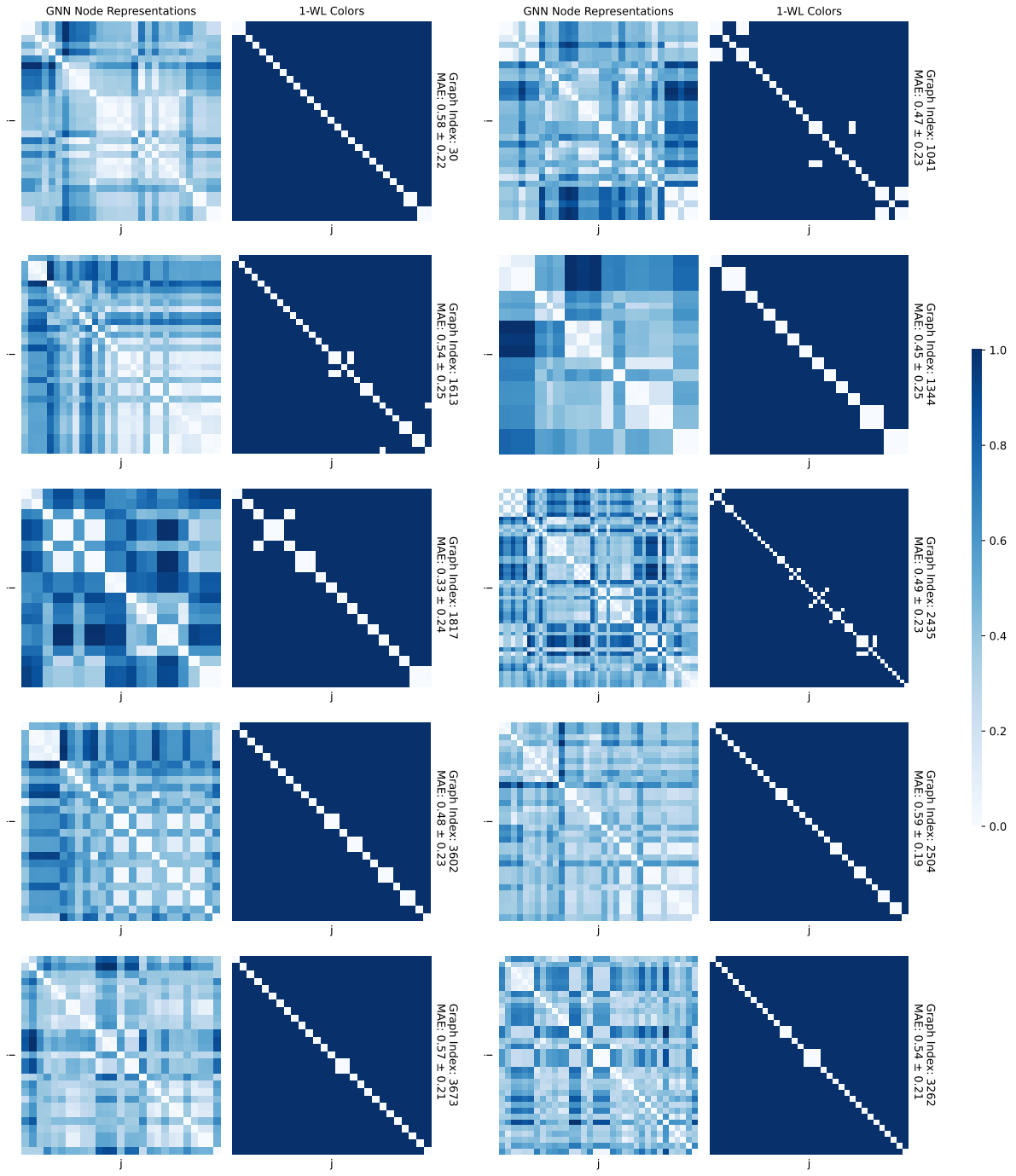


Figure B.16.: Visualizing the performance of the best performing GNN on the NCI1 dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN's test set, and the 1-WL algorithm ran only for three iteration. The average error for the entire test set is 0.50 ± 0.24.

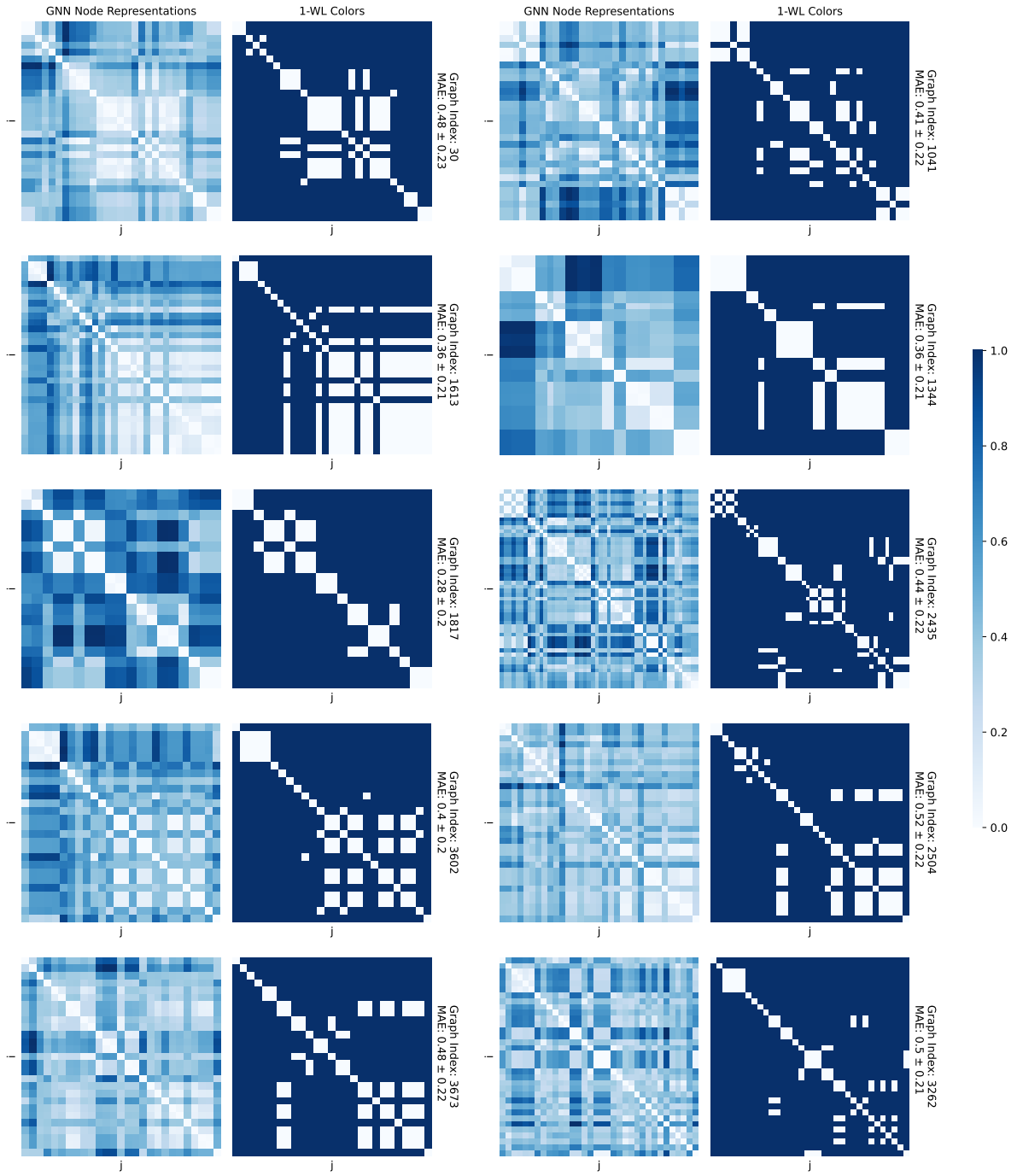


Figure B.17.: Visualizing the performance of the best performing GNN on the NCI1 dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN's test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is 0.42 ± 0.22.

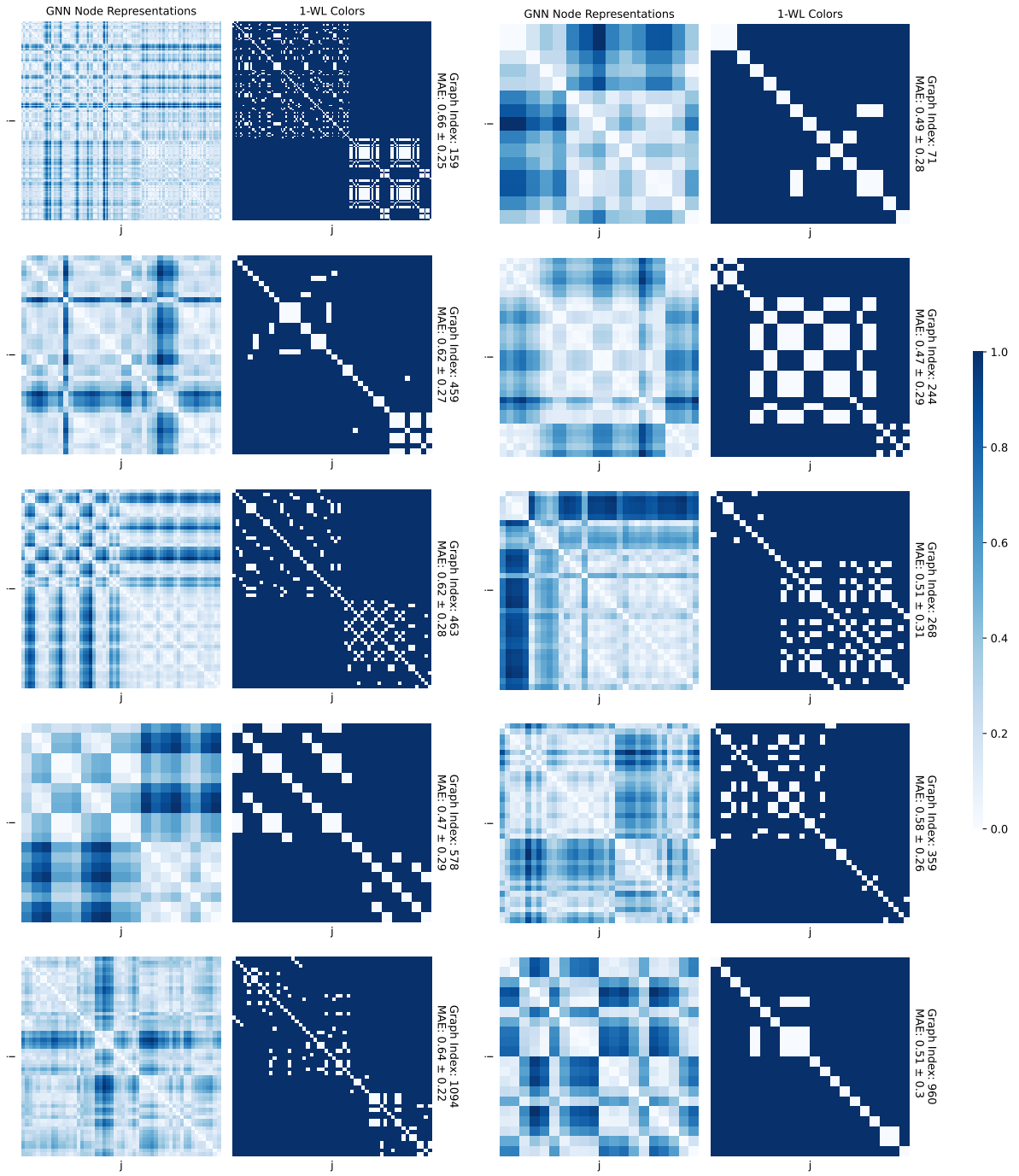


Figure B.18.: Visualizing the performance of the best performing GNN on the PROTEINS dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN's test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is 0.49 ± 0.26.

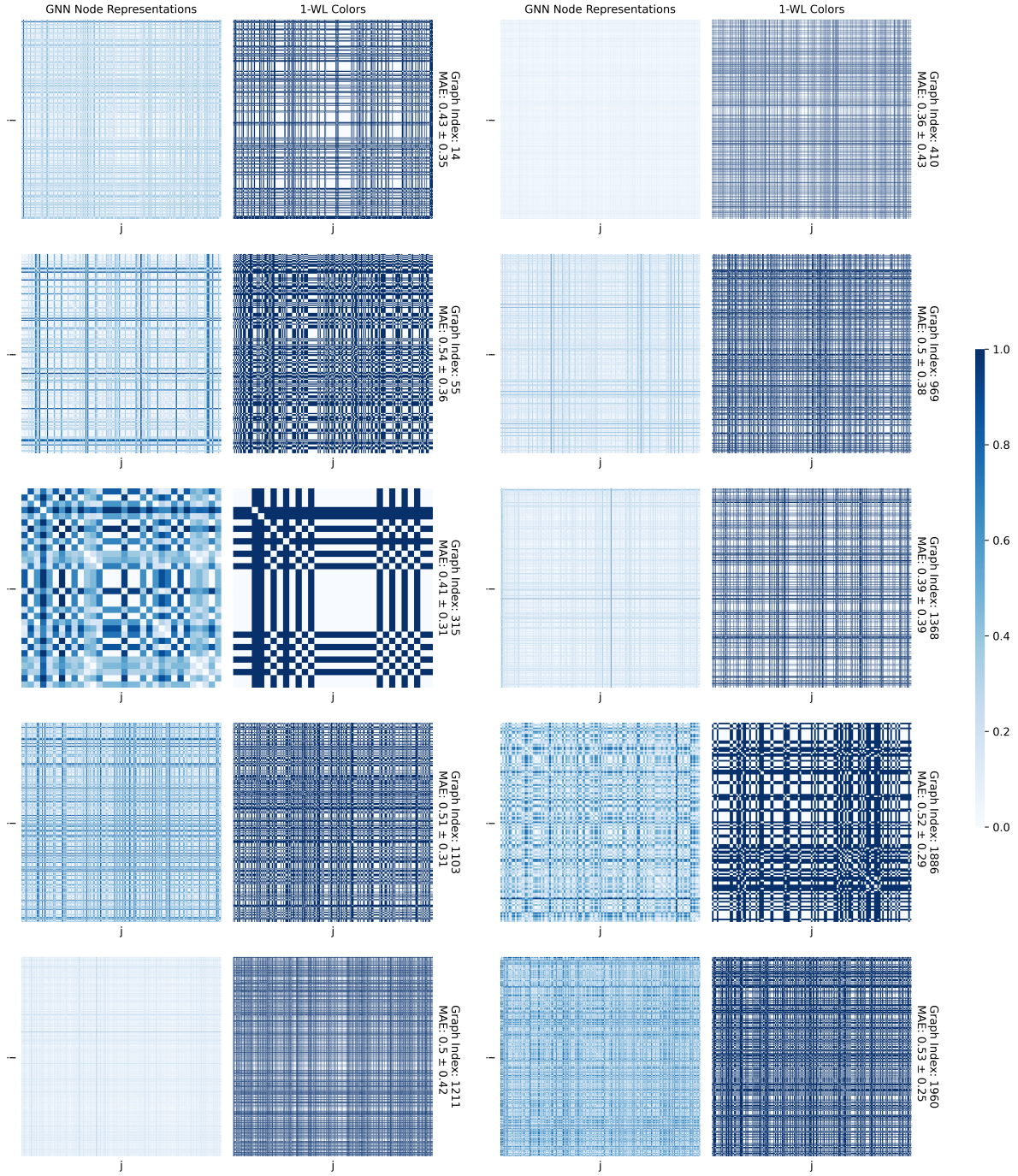


Figure B.19.: Visualizing the performance of the best performing GNN on the REDDIT-BINARY dataset in approximating node colors computed by the 1-WL algorithm. The ten graphs shown are randomly sampled from the GNN's test set, and the 1-WL algorithm ran only for one iteration. The average error for the entire test set is 0.48 ± 0.32.