Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Graph neural networks and extensibility of partial graph automorphisms

Bachelor Thesis

2026
Samuel Varchol

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Graph neural networks and extensibility of partial graph automorphisms

Bachelor Thesis

Study Programme:	Applied Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	Mgr. Ján Pastorek

Bratislava, 2026
Samuel Varchol

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Samuel Varchol
Study programme:	Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Graph neural networks and extensibility of partial graph automorphisms

Annotation: Graph neural networks (GNNs) are neural network architectures used for machine learning on graphs. Among others, graph neural networks are one of the main building blocks of AlphaFold, an artificial intelligence program developed by Google's DeepMind for solving the protein folding problem in biology.

Aim: The goal is to investigate potential applications of graph neural networks (GNNs) to problems in algebraic graph theory. The student will have the opportunity to engage with the state-of-art research in machine learning and algebraic graph theory and contribute to the field by training GNNs to predict algebraic properties.

Literature:
Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019). Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. _Proceedings of the AAAI Conference on Artificial Intelligence_, _33_(01), 4602–4609. [https://doi.org/10/ggfn97] (https://doi.org/10/ggfn97)

Paolino, R., Maskey, S., Welke, P., & Kutyniok, G. (2024). _Weisfeiler and Leman Go Loopy: A New Hierarchy for Graph Representational Learning_.

Wu, L., Cui, P., Pei, J., & Zhao, L. (Eds.). (2022). _Graph Neural Networks: Foundations, Frontiers, and Applications_. Springer Nature Singapore. [https://doi.org/10.1007/978-981-16-6054-2] (https://doi.org/10.1007/978-981-16-6054-2)

Supervisor:	Mgr. Ján Pastorek
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	25.02.2025
Approved:	28.04.2026

doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

Student

Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Samuel Varchol
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Graph neural networks and extensibility of partial graph automorphisms
Grafové neurónové siete a rozšíriteľnosť čiastočných automorfizmov

Anotácia:

Cieľ: Cieľom je preskúmať potenciálne aplikácie grafových neurónových sietí (GNN) na riešenie problémov algebraickej teórie grafov. Študent bude mať možnosť zapojiť sa do najmodernejšieho výskumu v oblasti strojového učenia a algebraickej teórie grafov a prispieť k tejto oblasti trénovaním GNN na predikciu algebraických vlastností.

Literatúra:

Vedúci: Mgr. Ján Pastorek
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 25.02.2025

Dátum schválenia: 28.04.2026 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Acknowledgments: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstract

Graphs encode their symmetries in the automorphism group, the set of vertex relabelings that preserve adjacency. A partial automorphism is a mapping defined on only some of the vertices that locally respects adjacency, and the question of whether such a partial mapping extends to a full automorphism is the partial automorphism extension problem. Although this problem reduces to graph isomorphism and is solved exactly by classical algorithms, it offers a clean benchmark for the expressive power of graph neural networks, because its hard instances coincide precisely with the configurations on which standard message passing collapses information. This thesis studies how well two contemporary graph neural network architectures can decide partial automorphism extension. We build a labeled benchmark of roughly 82,000 training and 6,670 test instances from the House of Graphs database, where hard negative examples are produced by a pseudo-similarity construction so that non-extensibility holds by design rather than by post-hoc filtering. We then train and analyze two models: a baseline message-passing network whose expressive power matches the limit of one-dimensional color refinement, and a hybrid architecture that combines message passing with global self-attention and spectral positional encodings. The baseline reaches 80.46% test accuracy and its ceiling proves robust both to added structural features and to a 1.5× increase in training data, isolating expressive power as the bottleneck rather than feature quality or data volume. The hybrid model reaches 97.77% accuracy, a roughly 89% reduction in error rate. Paired structural diagnostics locate the remaining errors: the baseline fails uniformly across all graphs that admit any non-trivial symmetry, while the residual errors of the hybrid model concentrate disproportionately on regular graphs, a pattern whose deeper structural cause remains open. The relative distance between original and mapped vertices, normalized by the graph diameter, emerges as the structural signal that best predicts where the more expressive model gains over the baseline.

Keywords: symmetry, partial automorphism extension, Weisfeiler-Lehman, graph neural networks

Abstrakt

Grafy kódujú svoje symetrie v grupe automorfizmov, teda v množine preoznačení vrcholov, ktoré zachovávajú susednosť. Čiastočný automorfizmus je zobrazenie definované iba na niektorých vrcholoch, ktoré lokálne rešpektuje susednosť, a otázka, či sa takéto čiastočné zobrazenie dá rozšíriť na úplný automorfizmus, sa nazýva problém rozšírenia čiastočného automorfizmu. Hoci sa tento problém dá redukovať na izomorfizmus grafov a klasické algoritmy ho riešia presne, predstavuje vhodný benchmark pre expresívnu silu grafových neurónových sietí, pretože jeho náročné inštancie presne zodpovedajú konfiguráciám, pri ktorých štandardné metódy typu message passing strácajú schopnosť rozlišovať informáciu. Táto práca skúma, ako dobre dokážu dve súčasné architektúry grafových neurónových sietí rozhodovať problém rozšírenia čiastočného automorfizmu. Vytvárame anotovaný benchmark pozostávajúci približne z 82 000 trénovacích a 6 670 testovacích inštancií z databázy House of Graphs, kde náročné negatívne príklady vznikajú pomocou konštrukcie pseudo-podobnosti, takže nerozšíriteľnosť je zaručená už samotnou konštrukciou, a nie dodatočným filtrovaním. Následne trénujeme a analyzujeme dva modely: základnú message-passing sieť, ktorej expresívna sila zodpovedá hranici jednorozmerného zjemňovania farieb, a hybridnú architektúru kombinujúcu message passing s globálnou self-attention a spektrálnymi pozičnými kódovaniami. Základný model dosahuje presnosť 80,46% na testovacej množine a jeho výkonnostný strop zostáva stabilný aj po pridaní štrukturálnych príznakov a po 1,5× zväčšení trénovacích dát, čo ukazuje, že hlavným obmedzením je expresívna sila modelu, nie kvalita príznakov ani objem dát. Hybridný model dosahuje presnosť 97,77%, čo predstavuje približne 89% zníženie chybovosti. Párové štrukturálne diagnostiky lokalizujú zostávajúce chyby: základný model zlyháva rovnomerne naprieč všetkými grafmi, ktoré pripúšťajú netriviálnu symetriu, zatiaľ čo zvyškové chyby hybridného modelu sa neúmerne koncentrujú na regulárnych grafoch, pričom hlbšia štrukturálna príčina tohto javu zostáva otvorená. Relatívna vzdialenosť medzi pôvodnými a zobrazenými vrcholmi, normalizovaná priemerom grafu, sa ukazuje ako štrukturálny signál, ktorý najlepšie predpovedá, kde expresívnejší model získava oproti základnému modelu.

Kľúčové slová: symetria, rozšíriteľnosť čiastočného automorfizmu, Weisfeiler-Lehman, grafové neurónové siete

Contents

Introduction

Over the last decade, machine learning has become an active tool for reasoning about discrete mathematical structures. Examples span the spectrum: graph neural networks are now a standard component of molecular property prediction and combinatorial optimization pipelines; supervised graph models have been trained to predict Boolean satisfiability directly from formula graphs, with single-bit supervision recovering a working SAT classifier [34]; the expressive limits of message-passing GNNs have been tested via specific structural decision tasks such as substructure counting [8]; and reinforcement learning has been used to search for explicit counterexamples to conjectures in extremal graph theory [39, 14]. A common thread is that the underlying objects carry rich combinatorial structure which classical algorithms address with worst-case guarantees but limited flexibility, while learned models trade exactness for the ability to generalize across instances and to exploit empirical regularities.

This thesis applies the same trend to a problem from algebraic graph theory. Graphs encode their symmetries in the automorphism group $\mathrm{Aut}(G)$, which describes how a graph can be relabeled without changing its structure. In many settings only a partial view of this symmetry is available: a candidate mapping $f : V_1 \to V_2$ defined on two (not necessarily identical) subsets of the vertices that locally looks like an automorphism, and the natural question is whether f is the restriction of a genuine global symmetry. This is the partial automorphism extension problem, formalized in Chapter 2, and it is the focus of this thesis.

The extension question is algorithmically tractable. It reduces to a graph isomorphism test on a suitably colored copy of G, and modern implementations such as McKay's `nauty` [28], accessed in our pipeline through the Python binding `pynauty`, decide it exactly. What makes the problem interesting from a learning perspective is not the absence of a solver but the structure of its hard instances. Extensibility is a global property: a partial map can be locally valid, preserving every adjacency constraint among mapped vertices, yet still be non-extensible because the vertices involved lie in different orbits of $\mathrm{Aut}(G)$. The witnesses to non-extensibility are precisely the configurations that local procedures cannot separate: pseudo-similar vertex pairs, regular substructures, and other instances on which the 1-dimensional Weisfeiler-Leman test (1-WL) collapses information that the global automorphism group keeps apart. In

the longer term, a model that learns where these hard configurations lie could serve as a search heuristic inside the backtracking solver itself, analogously to how learned branching heuristics have improved CDCL SAT solvers [26].

A benchmark for GNN expressiveness. This thesis uses partial automorphism extension as a benchmark for the expressive power of graph neural networks. Standard message-passing GNNs are upper-bounded by 1-WL [42, 29], and the extension problem is hard exactly on the configurations 1-WL cannot separate, so the task aligns the empirical performance ceiling of a model with a theoretical hardness characterization rather than with a dataset artifact. Three properties make this benchmark well-suited to the question: (i) the label is binary and exactly defined, with no annotation noise; (ii) the difficulty of an instance has a structural meaning (the WL-equivalence class of the witnessing vertex configuration) that can be measured graph by graph; and (iii) negative examples can be *constructed* to be cross-orbit by design, via a Godsil–Kocay pseudo-similarity construction and a cross-orbit blocking step, so the negative class isolates the genuinely hard configurations rather than easy adjacency violations.

Research question. How well can contemporary graph neural network architectures decide the partial automorphism extension problem, and what structural properties of the input explain where they succeed or fail? The investigation runs along a single axis. We first measure how far a 1-WL-aligned baseline (the Graph Isomorphism Network, GIN) can go on this task, and we then ask how much of the remaining gap a more expressive architecture (GraphGPS with Laplacian positional encodings [33]) can close, and what structural property of (G, f) explains the gap.

Thesis outline. The prerequisites are collected in Chapter 1: graph theory and partial automorphisms (Section 1.1), the Weisfeiler-Leman algorithm (Section 1.2), and message-passing GNNs including GIN (Section 1.3). On that foundation, Chapter 2 formalizes the partial automorphism extension problem as binary classification, describes the dataset construction (with the pseudo-similar and blocking strategies for negatives), and defines the vertex features used by the models. The GIN baseline experiments, together with a diagnosis of where and why a 1-WL-bounded architecture fails, are reported in Chapter 3. Chapter 4 then introduces GraphGPS, presents the headline result, and isolates the structural properties of (G, f) that explain the gap between the two models. The Conclusion summarizes the findings and lists open questions.

Chapter 1

Preliminaries

This chapter collects the background needed to formulate and study the main problem of this thesis. Section 1.1 introduces the relevant concepts from graph theory, finishing with the notions of partial automorphisms and their extensibility, which are the central objects we aim to classify. Section 1.2 describes the Weisfeiler-Leman color refinement algorithm, a classical graph isomorphism heuristic whose expressive power determines what graph neural networks can and cannot learn. Section 1.3 then introduces message-passing neural networks and the Graph Isomorphism Network (GIN), the primary architecture used in this thesis, and explains how the 1-WL expressiveness ceiling constrains the ability of standard GNNs to capture graph symmetries.

1.1 Graphs and symmetries

In this section, we summarize the foundational concepts of graph theory and review some algebraic aspects of graphs that we will need to formulate a task for graph neural networks.

1.1.1 Introduction to graph theory

This subsection follows definitions and concepts from the textbooks by Diestel [10] and West [41].

Formally, a *graph* G is defined as a pair $G = (V, E)$, where:

- V is a finite set of *vertices* (or *nodes*)
- E is a set of *edges*, that is, unordered pairs of distinct vertices

Throughout this thesis we consider only *simple* graphs: finite, undirected, and without self-loops or multiple edges between the same pair of vertices. The vertex set of a graph G is referred to as $V(G)$ and the edge set as $E(G)$. Two vertices $u, v \in V(G)$ are *adjacent* if $\{u, v\} \in E(G)$.

The *neighborhood* of a vertex $v \in V(G)$ is the set of all vertices adjacent to v:

$$N(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}.$$

The *degree* of a vertex v, denoted $\deg(v)$, is the number of its neighbors, i.e., $\deg(v) = |N(v)|$.

When studying graphs, it is often useful to focus on a part of the graph rather than the whole. This leads to the notion of a subgraph.

A graph G' is a *subgraph* of graph G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$, where every edge in $E(G')$ has both endpoints in $V(G')$.

A particularly useful type of subgraph is one that retains all the edges between its vertices that were present in the original graph.

An *induced subgraph* $G' = (V', E')$ of a graph $G = (V, E)$ is a subgraph such that the edge set E' consists of all the edges in E that connect vertices in V'. Formally,

$$E' = \{\{u, v\} \in E \mid u, v \in V'\}.$$

Induced subgraphs will reappear later when we define partial automorphisms.

Another fundamental aspect of graph structure is how vertices are connected and how far apart they can be. A *path* in G is a sequence of distinct vertices $v_0, v_1, \ldots, v_k$ such that $\{v_{i-1}, v_i\} \in E(G)$ for each $i = 1, \ldots, k$; its *length* is k. The *distance* $d_G(u, v)$ between two vertices $u, v \in V(G)$ is the length of a shortest path between them. The *diameter* of G, denoted $\operatorname{diam}(G)$, is the maximum distance over all pairs of vertices in G; it measures how "wide" the graph is. These notions will be used later when analyzing the reach of graph neural networks and the structure of graph symmetries.

1.1.2 Algebraic graph theory

In this subsection, we focus on the algebraic aspects of graphs, mainly on automorphisms. We follow the definitions from the textbook by Biggs [4].

Central to this thesis is the concept of symmetry in graphs: which vertices are structurally interchangeable, and to what extent can such symmetries be extended? Algebraic graph theory provides the tools to make these questions precise.

An *isomorphism* between two graphs G_1 and G_2 is a bijection $f : V(G_1) \to V(G_2)$ such that $\{u, v\} \in E(G_1)$ if and only if $\{f(u), f(v)\} \in E(G_2)$. If such a bijection exists, we say that the graph G_1 is *isomorphic* to G_2, written as $G_1 \cong G_2$. In other words, two graphs are isomorphic if they have the same structure.

A natural special case arises when we ask whether a graph is isomorphic to itself. This is called an automorphism.

An *automorphism* of a graph $G = (V, E)$ is a bijection $f : V \to V$ such that $\{f(u), f(v)\}$ is an edge of G if and only if $\{u, v\}$ is an edge of G. In other words, an automorphism is an isomorphism from graph G to itself.

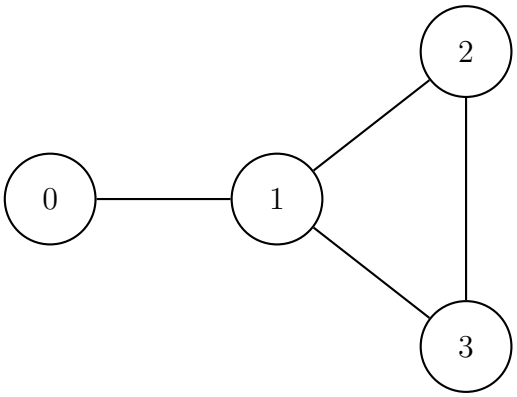


Figure 1.1: Example graph with a non-trivial automorphism.

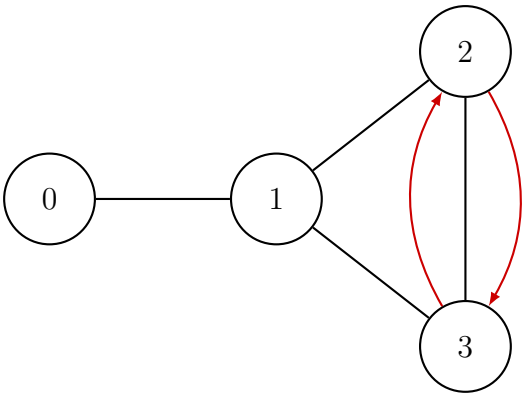


Figure 1.2: Mapping of vertices under the non-trivial automorphism f. Red arrows show f.

The set of all automorphisms of a graph G is called the *automorphism group* of G, denoted by $\mathrm{Aut}(G)$. The *order* of the automorphism group is the number of its elements, denoted $|\mathrm{Aut}(G)|$.

The *trivial automorphism* is the identity automorphism, which maps every vertex to itself.

Graphs with non-trivial automorphisms are of particular interest, as we want to see if graph neural networks can learn to solve tasks related to graph symmetries. The following example illustrates such a graph.

Let $G = (V, E)$ be a graph with vertex set $V = \{0, 1, 2, 3\}$ and edge set $E = \{\{0, 1\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$, as illustrated in Figure 1.1. One non-trivial automorphism of this graph is the permutation f defined as follows:

$$f(0) = 0, \quad f(1) = 1, \quad f(2) = 3, \quad f(3) = 2.$$

This permutation maps vertex 2 to vertex 3 and vertex 3 to vertex 2, while keeping vertices 0 and 1 fixed, as shown in Figure 1.2.

We can relax the requirement of a full automorphism by considering structure-preserving bijections between subsets of vertices rather than the entire vertex set. This yields the notion of a partial automorphism, which is the central object studied in this thesis.

Let $G[V_1]$ denote the subgraph of G induced by $V_1 \subseteq V(G)$. A *partial automorphism* of a graph $G = (V, E)$ is an isomorphism $f : G[V_1] \to G[V_2]$ between two induced subgraphs of G, where $V_1, V_2 \subseteq V(G)$ are non-empty; the sets V_1 and V_2 need not be equal. Equivalently, $f : V_1 \to V_2$ is a bijection such that for every pair u, v in V_1, $\{u, v\} \in E(G)$ if and only if $\{f(u), f(v)\} \in E(G)$. We say that a partial automorphism is non-trivial if $f(v) \neq v$ for at least one $v \in V_1$ [21]. We write $\mathrm{dom}(f) = V_1$ and $\mathrm{img}(f) = V_2$ for the domain and image of f.

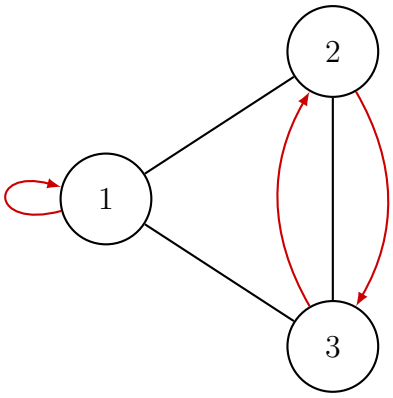


Figure 1.3: An extensible partial automorphism on the graph of Figure 1.1: $f(1) = 1,\ f(2) = 3,\ f(3) = 2$. It extends to the full automorphism of Figure 1.2 by setting $\phi(0) = 0$. Red arrows show f; self-loops mark fixed points.

A natural question is whether a given partial automorphism arises as the restriction of some full automorphism. We capture this by the following notion, which is the main focus of this thesis.

A partial automorphism $f : V_1 \to V_2$ of G is *extensible* if there exists a full automorphism $\phi \in \mathrm{Aut}(G)$ whose restriction to V_1 (the function obtained by applying ϕ only to vertices in V_1) coincides with f, that is, $\phi|_{V_1} = f$.

We illustrate both possibilities on the graph of Figure 1.1.

An extensible partial automorphism. Consider $f : \{1,2,3\} \to \{1,2,3\}$ with $f(1) = 1,\ f(2) = 3,\ f(3) = 2$, shown in Figure 1.3. This is a partial automorphism because the induced subgraph on $\{1,2,3\}$ is the triangle $\{\{1,2\},\{1,3\},\{2,3\}\}$, which f permutes to itself. It is extensible: setting $\phi(0) = 0$ and inheriting f on $\{1,2,3\}$ yields the full automorphism of Figure 1.2.

A non-extensible partial automorphism. Now consider $f : \{1,2,3\} \to \{1,2,3\}$ with $f(1) = 2,\ f(2) = 1,\ f(3) = 3$, shown in Figure 1.4. Again f permutes the triangle on $\{1,2,3\}$ to itself, so it is a partial automorphism. But it is *not* extensible: the only unmapped vertex is 0, so any extending automorphism ϕ must satisfy $\phi(0) = 0$. Then $\{\phi(0), \phi(2)\} = \{0,1\} \in E(G)$ while $\{0,2\} \notin E(G)$, violating adjacency preservation (Figure 1.5).

The automorphisms of a graph naturally group its vertices: if some automorphism maps one vertex to another, the two vertices are structurally equivalent. This equivalence is captured by the notion of orbits.

The *orbit* of a vertex $v \in V(G)$ under the automorphism group $\mathrm{Aut}(G)$ is the set

$$\mathrm{orb}(v) = \{\sigma(v) \mid \sigma \in \mathrm{Aut}(G)\}.$$

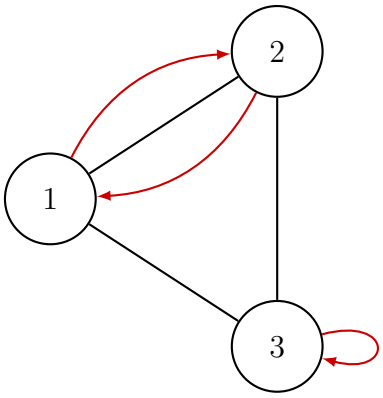


Figure 1.4: A partial automorphism on the graph of Figure 1.1: $f(1) = 2$, $f(2) = 1$, $f(3) = 3$. Red arrows show f; self-loops mark fixed points.

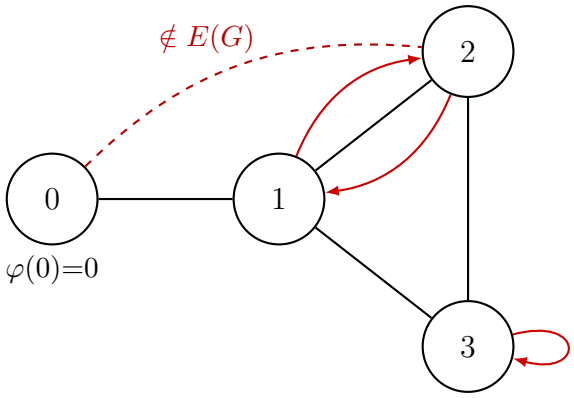


Figure 1.5: The failed extension of the partial automorphism in Figure 1.4: forcing $\phi(0) = 0$ gives $\{\phi(0), \phi(2)\} = \{0, 1\} \in E(G)$, yet the preimage pair $\{0, 2\} \notin E(G)$, violating adjacency preservation.

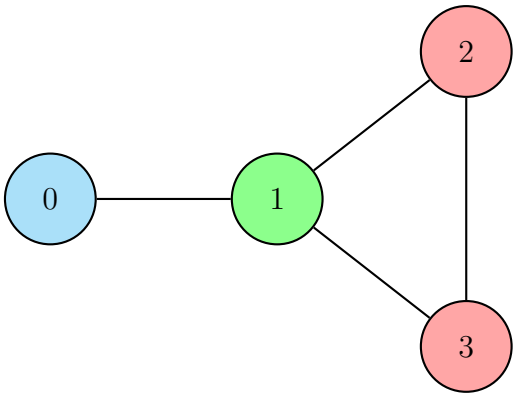


Figure 1.6: Orbit partition of the graph from Figure 1.1. Each color represents a distinct orbit: $\{0\}$, $\{1\}$, and $\{2, 3\}$.

The orbits of all vertices partition $V(G)$ into disjoint sets; this partition is called the *orbit partition* of G.

Such an orbit partition is illustrated in Figure 1.6 for the graph from Figure 1.1, where the vertices are colored according to their orbits.

In this example, vertices 0 and 1 are in their own orbits, while vertices 2 and 3 belong to the same orbit. Since every automorphism must map each vertex to a vertex in the same orbit, any automorphism must fix 0 and 1 and permute $\{2, 3\}$. The only two options are the identity and the automorphism swapping 2 and 3; this means that the graph has exactly one non-trivial automorphism.

To state the notion of pseudo-similar vertices, we first recall vertex deletion. For a vertex $u \in V(G)$, we write $G - u$ for the subgraph of G induced by $V(G) \setminus \{u\}$, i.e. the graph obtained by deleting u and all edges incident to it.

A subtle counterpart to orbit equivalence is the notion of *pseudo-similar vertices*, introduced by Kimble, Schwenk, and Stockmeyer [23]. Two vertices $u, v \in V(G)$ are pseudo-similar if $G - u \cong G - v$ but no automorphism of G maps u to v; equivalently, u and v lie in distinct orbits of $\mathrm{Aut}(G)$ yet are indistinguishable once either is deleted.

An example of pseudo-similar vertices is given by the graph in Figure 1.7, where vertices 5 and 7 are pseudo-similar: if we delete either vertex (Figure 1.8 or Figure 1.9), the resulting graphs are isomorphic but no automorphism of the original graph maps 5 to 7.

Pseudo-similar pairs are precisely the situations in which local structure agrees, but global symmetry does not. Godsil and Kocay [16] showed that all graphs with such pairs can be constructed this way; we use their construction to manufacture non-extensible negative examples in Chapter 2.

We will use these concepts from graph theory in the following chapters, where we study graph neural networks and use them to predict the extensibility of partial automorphisms.

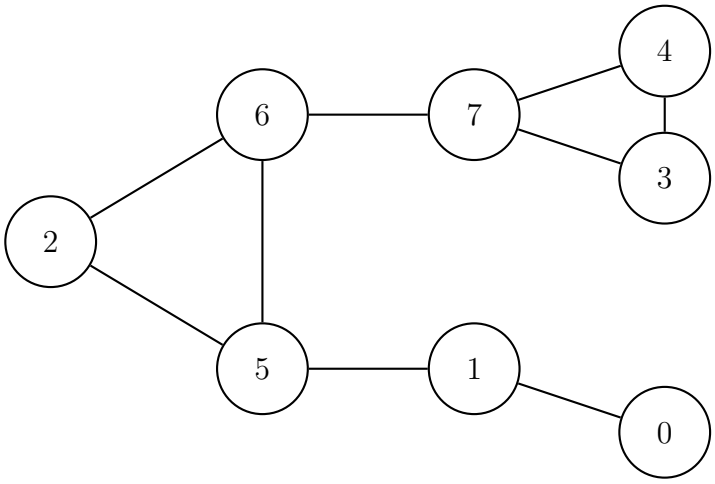


Figure 1.7: A graph with pseudo-similar vertices 5 and 7.

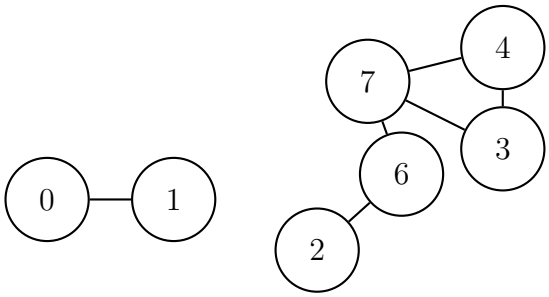


Figure 1.8: Graph obtained by deleting vertex 5 from the graph of Figure 1.7.

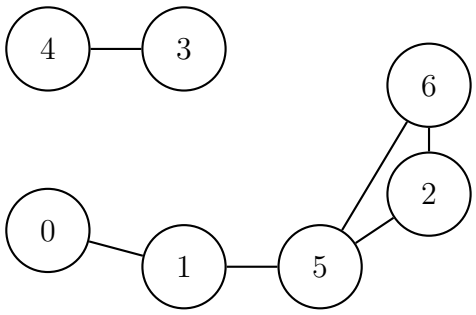


Figure 1.9: Graph obtained by deleting vertex 7 from the graph of Figure 1.7.

1.2 The Weisfeiler-Leman algorithm

The Weisfeiler-Leman (WL) algorithm is a classical heuristic for the graph isomorphism problem. It is closely connected to the expressiveness (the ability to distinguish non-isomorphic graphs) of graph neural networks (GNNs). Understanding the WL algorithm is essential before studying GNNs.

This section describes the algorithm, its color refinement procedure, and the connection to GNN expressiveness established by Morris et al. [29].

We follow the works of Huang and Villar [19] and Shervashidze et al. [36].

1.2.1 Graph isomorphism testing

There are many graph similarity measures, but the most natural is to check if two graphs are structurally identical, that is, if they are isomorphic. Ideally, we want to have a binary test that returns true if two graphs are isomorphic and false if they are not.

Graph isomorphism testing has practical applications in areas such as mathematical chemistry, where molecular graphs are matched against databases [20], and electronic design automation, where circuit diagrams are verified against layouts [3].

The graph isomorphism problem is known to be in NP, but it is one of the few problems that has no classification as either polynomial-time solvable or NP-complete. The best known algorithm, due to Babai [2], runs in quasi-polynomial[1] time $2^{O((\log n)^3)}$, but no polynomial-time algorithm is known for the general case.

One such heuristic is the Weisfeiler-Leman algorithm, first described by B. Weisfeiler and A.A. Leman in 1968 [40].

1.2.2 1-dimensional Weisfeiler-Leman

We will focus on the 1-dimensional variant of the algorithm, which is the simplest version of the Weisfeiler-Leman algorithm and the most relevant to the study of graph neural networks.

The 1-dimensional Weisfeiler-Leman algorithm (1-WL), also known as "color refinement", takes a graph as input and returns a multiset of vertex colors. A *multiset* differs from an ordinary set in that it allows repeated elements: $\{\{a, a, b\}\}$ is a valid multiset in which a appears twice. Each vertex is assigned an initial color, and the coloring is iteratively refined from neighbor colors until it stabilizes; the multiset of colors at stabilization is the algorithm's output. We describe the procedure in detail in Subsection 1.2.3.

[1] Slower than any polynomial but faster than exponential.

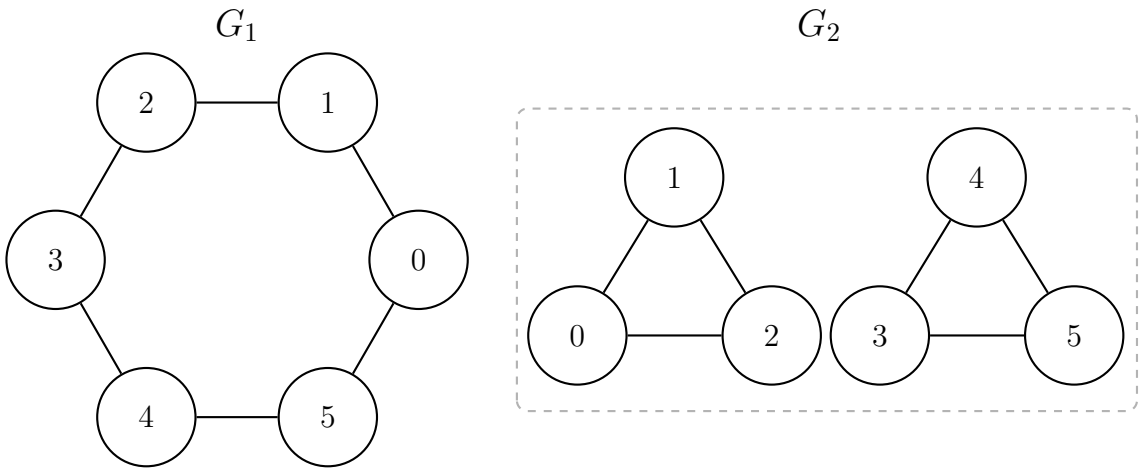


Figure 1.10: Two non-isomorphic 2-regular graphs (every vertex has degree 2): one connected (G_1) and one consisting of two disjoint triangles (G_2). Because every vertex has the same degree and the same neighborhood multiset after any number of refinement steps, 1-WL assigns identical colors to all vertices in both graphs and cannot distinguish them: the stabilized color multiset is $\{\{c, c, c, c, c, c\}\}$ (a single color c for all six vertices) for both G_1 and G_2.

1-WL is used as a heuristic for graph isomorphism testing. If two graphs are isomorphic, their output color multisets are identical; the converse, however, does not hold in general [19]: two non-isomorphic graphs may still produce identical color multisets, as Figure 1.10 shows.

1-WL can be generalized to the k-WL algorithm, where the labels are given to k-tuples of vertices instead of single vertices. This leads to a more powerful algorithm that can distinguish more graphs than 1-WL, although it is also more computationally expensive [29]. Cai, Fürer, and Immerman [6] showed that for every k there exist non-isomorphic graphs that k-WL cannot distinguish, so the hierarchy is strict: no fixed k suffices to decide graph isomorphism in general.

1.2.3 The 1-WL color refinement procedure

The 1-WL is summarized as:

Initialization: If the graph has vertex features, vertices are initialized with their feature labels; otherwise all vertices receive the same initial color $c^{(0)}$ (see Figure 1.11).

Refinement step: At each iteration t, the color of vertex v is updated as:

$$c^{(t+1)}(v) = \mathrm{HASH}\big(c^{(t)}(v),\ \{\{c^{(t)}(u) : u \in N(v)\}\}\big) \tag{1.1}$$

where $\{\{\cdot\}\}$ denotes a multiset and HASH is an injective function on the pair (current color, multiset of neighbor colors); injectivity is what guarantees that distinct inputs yield distinct refined colors, and it is precisely the property that the GIN architecture

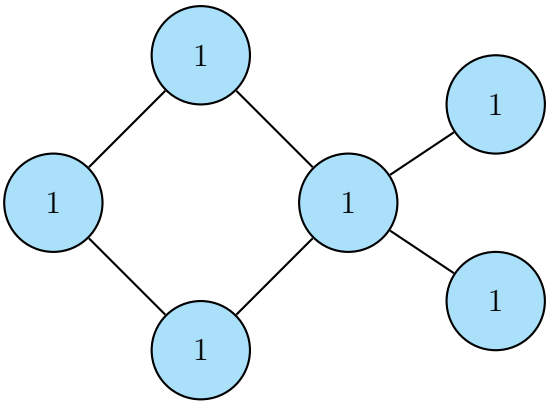


Figure 1.11: Initialization step of the 1-WL algorithm. All vertices are assigned the same initial color.

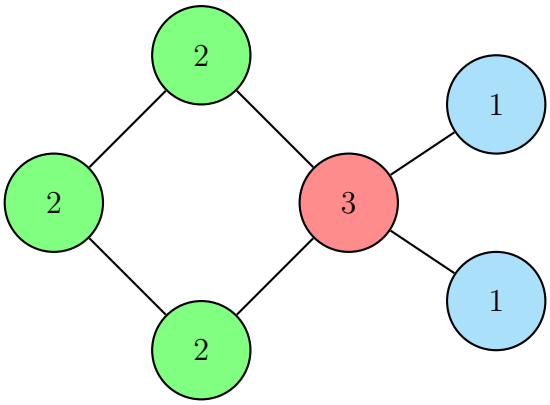


Figure 1.12: After the first refinement step, vertices with different degrees are separated.

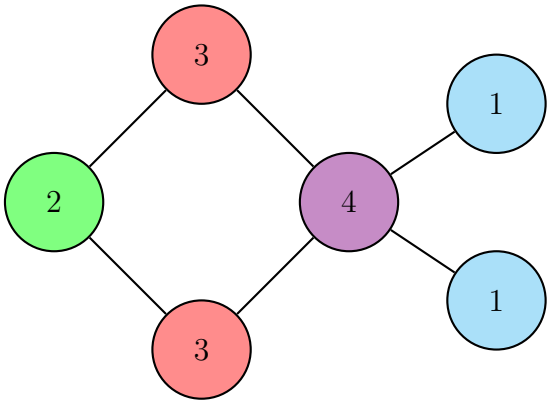


Figure 1.13: After the second refinement step, the coloring stabilizes and does not change anymore.

in Subsection 1.3.2 reproduces in a learnable way. Two vertices v, w receive different colors at step $t+1$ whenever they differ in their current color or in the multiset of neighbor colors. Refinement is repeated until the coloring stabilizes, meaning no colors changed in the last iteration (see Figures 1.12 and 1.13).

Since at most $|V|$ distinct colors can exist and the partition of vertices only becomes finer over time, the algorithm is guaranteed to terminate in at most $|V|-1$ iterations [29]. Here *finer* means that any two vertices that differ in color at step t remain in different classes at step $t+1$. Each strict refinement increases the number of color classes by at least one, and the number of classes is bounded by $|V|$, which gives the $|V|-1$ bound. A naive implementation therefore runs in $O(|V| \cdot (|V|+|E|))$ time. Using the partition-refinement technique of Paige and Tarjan [31], originally due to Cardon and Crochemore [7], the total runtime can be improved to $O\big((|V|+|E|)\log|V|\big)$, independent of the number of iterations.

We denote the stabilized color multiset by 1-WL(G). We say that 1-WL was suc-

cessful at distinguishing a pair of non-isomorphic graphs G, G' if 1-WL$(G) \neq$ 1-WL(G'). However, if 1-WL$(G) =$ 1-WL(G'), we cannot conclude whether G and G' are isomorphic or non-isomorphic.

Remark. The 1-WL coloring is related to the orbit partition introduced in Section 1.1: every automorphism of a graph preserves the WL coloring, so vertices in the same orbit always receive the same color. This does not hold in the reverse direction, as vertices with the same WL color may belong to different orbits. This is precisely the limitation that carries over to graph neural networks.

1.2.4 Connection to graph neural networks

The 1-WL algorithm has a direct analogue in message-passing GNNs: a message-passing GNN replaces the injective HASH function with a learned neural network and replaces multiset aggregation over neighbor colors with a differentiable aggregation over neighbor feature vectors. As a result, the expressive power of standard message-passing GNNs is upper-bounded by the 1-WL test. This means that if two graphs are not distinguishable by 1-WL, neither is any standard GNN. Section 1.3 makes this connection precise.

1.3 Graph neural networks

We now introduce message-passing neural networks (MPNNs) as the learnable analogue of the 1-WL color refinement procedure, and then describe the Graph Isomorphism Network (GIN) by Xu et al. [42], which provably attains the 1-WL upper bound.

The broader motivation for graph-based deep learning is the prevalence of graph-structured data: social networks, physical systems, protein–protein interaction networks, and knowledge graphs all carry rich relational information that flat feature vectors cannot capture. Earlier deep learning breakthroughs relied on convolutional neural networks (CNNs), which exploit local connectivity, weight sharing, and deep stacks of local filters on regular Euclidean grids such as images. Extending these ideas to non-Euclidean, irregular graph domains is the program known as *geometric deep learning* [5]; the name reflects the unifying view that graphs carry an underlying geometry, in contrast to the flat Euclidean grid assumed by CNNs.

1.3.1 Neural networks on graphs

A graph neural network is a function that maps an input graph to a vector representation of each of its vertices in some Euclidean space $\mathbb{R}^d$, where the embedding dimension

d is a chosen hyperparameter. Graph-level representations can then be derived by aggregating the vertex embeddings. Unlike images or sequences, which fit into fixed-size arrays, graphs vary in size: one graph may have a handful of vertices while another has millions, yet a practical model must process both with the same fixed set of parameters.

The term *non-Euclidean* refers to the combinatorial domain of the input: a graph has no fixed grid, no vertex coordinates, and no metric inherited from any ambient space; its structure is encoded entirely by the edge set. The initial vertex features $\mathbf{h}_v^{(0)} \in \mathbb{R}^d$ are Euclidean attribute vectors attached to vertices, but they carry no information about how vertices relate to one another. The role of the GNN is precisely to fuse these local features with the relational structure of the graph and produce output embeddings in $\mathbb{R}^d$.

We assume each vertex carries such an attribute vector, for example a one-hot encoding of a discrete vertex type or a learned embedding. When no attributes are given, every vertex receives the same constant feature, mirroring the uniform initialization of 1-WL in Section 1.2.

A fundamental property of most graph neural networks is *permutation equivariance* at the vertex level and *permutation invariance* at the graph level. Intuitively, the learned representation of the graph should not depend on how we choose to label its vertices: relabeling the vertices first and then applying the GNN should give the same result as applying the GNN first and then relabeling the outputs (for vertex-level models), or simply the same output (for graph-level models).

Concretely, let G be a graph on n vertices with adjacency matrix $A \in \{0,1\}^{n\times n}$ and stacked vertex features $X \in \mathbb{R}^{n\times d}$, and let $P \in \{0,1\}^{n\times n}$ be a permutation matrix representing a relabeling of the vertices. A vertex-level GNN $\mathrm{GNN}_V : (A, X) \mapsto H \in \mathbb{R}^{n\times d'}$ is permutation *equivariant* if

$$\mathrm{GNN}_V(PAP^\top,\, PX) \;=\; P\,\mathrm{GNN}_V(A, X)$$

for every permutation P, and a graph-level model $\mathrm{GNN}_G : (A, X) \mapsto \mathbf{h}_G \in \mathbb{R}^{d'}$ is permutation *invariant* if

$$\mathrm{GNN}_G(PAP^\top,\, PX) \;=\; \mathrm{GNN}_G(A, X).$$

These properties, together with the locality of message passing, confine standard MPNNs to the 1-WL expressiveness ceiling, as Subsection 1.3.2 makes precise.

The dominant GNN architecture is the *message-passing neural network* (MPNN) [32], originally formulated by Gilmer et al. [15] for learning molecular properties. It has since become the foundation for many subsequent GNN architectures because of its effectiveness across vertex, edge, and graph-level tasks [45].

The MPNN forward pass has two phases: *message passing* and *readout*. At each layer k, every vertex v first aggregates the representations of its neighbors via a

permutation-invariant AGGREGATE$^{(k)}$ function, and then combines the result with its own current representation via COMBINE$^{(k)}$ [42]:

$$\mathbf{a}_v^{(k)} = \text{AGGREGATE}^{(k)}\big(\{\{\mathbf{h}_u^{(k-1)} : u \in N(v)\}\}\big)$$
$$\mathbf{h}_v^{(k)} = \text{COMBINE}^{(k)}\big(\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)}\big)$$

where $\mathbf{h}_v^{(k)} \in \mathbb{R}^d$ is the feature vector of vertex v at layer k, with $\mathbf{h}_v^{(0)}$ being the initial vertex features, and $\mathbf{a}_v^{(k)}$ is the aggregated neighborhood message at layer k.

Each layer performs exactly one step of neighborhood expansion in the graph. The embedding $\mathbf{h}_v^{(k)}$ therefore captures the rooted subgraph of depth k centered at v: at layer 0 only v's own initial features are present; after layer 1, information from its direct neighbors is folded in; after layer k, every vertex reachable within k hops has contributed. With K layers, the receptive field of each vertex is its K-hop neighborhood.

The choice of AGGREGATE$^{(k)}$ directly affects expressiveness; the GIN model (Subsection 1.3.2) picks sum for this reason. After K layers, for graph-level tasks a READOUT function aggregates the final vertex representations into a single graph embedding:

$$\mathbf{h}_G = \text{READOUT}\big(\{\{\mathbf{h}_v^{(K)} \mid v \in V(G)\}\}\big)$$

The functions AGGREGATE$^{(k)}$, COMBINE$^{(k)}$, and READOUT are learned differentiable functions; different choices yield different GNN architectures.

Crucially, all vertices use the same message-passing and state-update functions, that is, the parameters of the neural network are shared across the graph. The total number of learnable parameters therefore depends only on the chosen functions and the number of layers K, never on $|V|$ or $|E|$. This is precisely what allows the same trained model to be applied to graphs of any size: the forward pass scales with the graph, but the parameter count does not. Weight sharing also, together with the permutation-invariant multiset aggregation, makes the learned function permutation-equivariant at the vertex level and permutation-invariant at the graph level [17].

This is the learnable analogue of the 1-WL procedure. The correspondence is direct: AGGREGATE$^{(k)}$ plays the role of collecting the multiset of neighbor colors, COMBINE$^{(k)}$ plays the role of the injective HASH that pairs a vertex's previous color with that multiset, and the vertex feature $\mathbf{h}_v^{(k)}$ takes the place of the discrete color $c^{(k)}(v)$. The difference is that 1-WL operates on a countable color alphabet with an exact injective HASH, whereas an MPNN operates on real-valued feature vectors with learned, only *approximately* injective AGGREGATE and COMBINE functions. As a

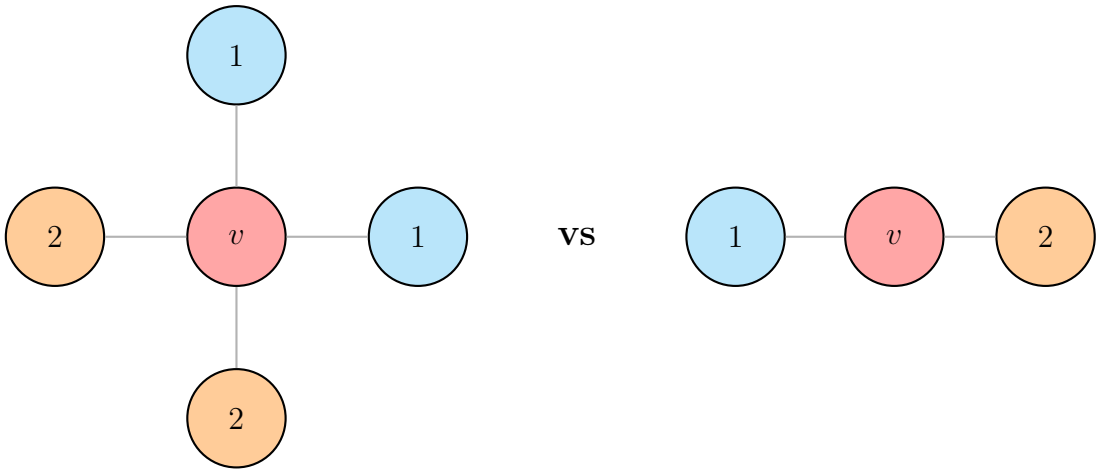


Figure 1.14: Mean-pool assigns identical embeddings to the red vertex v in both graphs: $\text{mean}(\{\{1, 1, 2, 2\}\}) = 1.5 = \text{mean}(\{\{1, 2\}\})$.

consequence, any pair of graphs that 1-WL cannot distinguish will also be indistinguishable to any standard MPNN [42].

Many concrete architectures, such as graph convolutional networks (GCN) [25], graph attention networks (GAT) [38], and others, instantiate this template with different choices of $\text{AGGREGATE}^{(k)}$, $\text{COMBINE}^{(k)}$, and READOUT, and have achieved strong results on many tasks. However, all share the 1-WL expressiveness ceiling.

The Graph Isomorphism Network, introduced next, is the variant that achieves the 1-WL bound exactly.

1.3.2 Graph Isomorphism Network (GIN)

The Graph Isomorphism Network (GIN), introduced by Xu et al. [42], is the MPNN architecture that provably matches 1-WL's expressive power while adding trainability. The central result of Xu et al. is that every MPNN is upper-bounded by 1-WL, and that this bound is tight: there exists an MPNN that matches 1-WL exactly, and GIN is that network.

Sum aggregation. The key idea is that the aggregation $\text{AGGREGATE}^{(k)}$ must be injective as a function on multisets in order to achieve 1-WL expressiveness [42]. Mean and max aggregation both fail this requirement:

- **Mean** cannot distinguish multisets that have the same proportion of each element but different total sizes, as shown in Figure 1.14.
- **Max** collapses any multiset to its largest element, losing multiplicity entirely, shown in Figure 1.15.

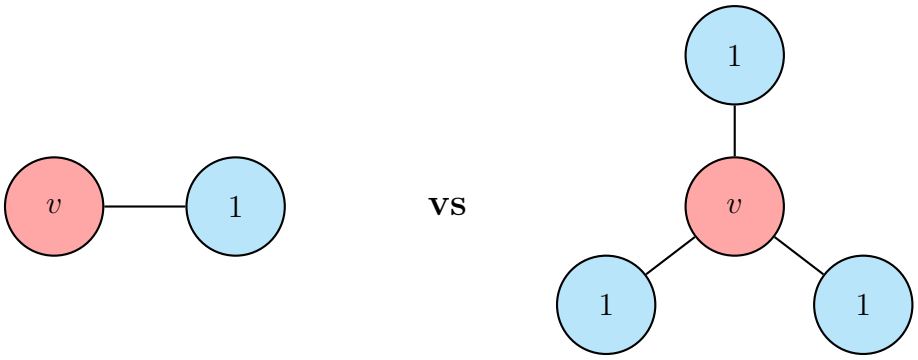


Figure 1.15: Max-pool assigns identical embeddings to the red vertex v in both graphs: $\max(\{\{1\}\}) = 1 = \max(\{\{1, 1, 1\}\})$.

Sum aggregation preserves both the identity and multiplicity of elements, and can therefore distinguish any two distinct multisets over a countable feature space [42].

Update rule. Each layer of GIN applies the following update to every vertex v:

$$\mathbf{h}_v^{(k)} = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)})\,\mathbf{h}_v^{(k-1)} + \sum_{u\in N(v)} \mathbf{h}_u^{(k-1)}\right) \tag{1.2}$$

where $\epsilon^{(k)}$ is a scalar that distinguishes the central vertex feature from its neighbor sum, and $\mathrm{MLP}^{(k)}$ is a multi-layer perceptron. Xu et al. analyze general values of $\epsilon^{(k)}$, but we use the variant with $\epsilon^{(k)} = 0$ ("GIN-0") in all experiments of this thesis. GIN-0 is known to match the full model on standard benchmarks [42].

Expressiveness. Xu et al. [42] show that GIN with a sufficient number of layers and a sufficiently expressive MLP is as powerful as 1-WL: it maps any two graphs that 1-WL distinguishes to different embeddings. Conversely, every MPNN (including GIN) fails on the same pairs that 1-WL fails on. This limitation is the learned counterpart of the orbit-partition remark from Section 1.2: vertices in the same 1-WL color class will receive identical GIN representations, regardless of the depth of the network.

Graph-level readout. To obtain a graph-level representation, GIN concatenates (denoted $\|$) the sum-pooled vertex representations from *all* layers, not just the last:

$$\mathbf{h}_G = \Big\|_{k=0}^{K} \sum_{v\in V(G)} \mathbf{h}_v^{(k)}.$$

The motivation is that a vertex representation at iteration k encodes its rooted subtree of depth k, the same structure that 1-WL refines at step k. Earlier layers therefore carry coarse, local subtree information and later layers carry deeper, more global information; concatenating the sums across all $K+1$ iterations ensures that features at every WL

depth contribute to the graph-level prediction, rather than only the deepest one [42]. This multiscale aggregation idea was introduced by Xu et al. in their earlier work on Jumping Knowledge Networks [43].

1.3.3 The 1-WL ceiling and paths beyond it

The upper bound established by Xu et al. [42] is not just a theoretical curiosity: it predicts concrete failures on any task that depends on distinguishing 1-WL-equivalent vertices or graphs. Regular and highly symmetric graphs contain vertices that 1-WL cannot distinguish, and therefore no standard MPNN (including GIN) can produce different representations for them, regardless of depth or width.

Three broad directions have been proposed to escape this ceiling:

- **Higher-order GNNs.** Operate on k-tuples of vertices rather than single vertices, matching the expressive power of k-WL at the cost of $O(n^k)$ memory [29].
- **Positional and structural encodings.** Augment vertex features with information that 1-WL cannot derive, such as Laplacian eigenvectors or random-walk landing probabilities [22]. This breaks the symmetry between 1-WL-equivalent vertices without changing the message-passing structure itself.
- **Global attention.** Replace or complement local aggregation with self-attention over all vertex pairs, giving every vertex a direct channel to every other vertex and removing the diameter-bounded information horizon of MPNNs [35].

The second and third directions combine naturally: encodings provide the raw symmetry-breaking information, and attention lets the model route that information globally.

The chapters that follow turn this theoretical picture into an empirical one. Chapter 2 formulates the partial automorphism extension problem as a supervised binary classification task and describes the dataset we use. Chapter 3 then trains GIN on this task as the strongest 1-WL-bounded baseline, and Chapter 4 returns to the escape routes of Subsection 1.3.3 by enriching GIN with GraphGPS [33], whose encodings and global attention are designed to break precisely the symmetries that defeat the baseline.

Chapter 2

Task formulation and dataset

In this chapter we formulate the partial automorphism extension problem as a supervised learning task, describe how we constructed our dataset of positive and negative examples, and define the vertex feature representations used by the models in Chapters 3 and 4.

2.1 Problem formulation

Intuitively, the partial automorphism extension problem asks whether a given partial automorphism of a graph can be extended to a full automorphism. We formalize this as a binary classification task: Given a graph $G = (V, E)$ and a partial automorphism $f : V_1 \to V_2$ (Subsection 1.1.2) of G, the task is to predict whether f is extensible, i.e. whether there exists $\phi \in \mathrm{Aut}(G)$ with $\phi|_{V_1} = f$. The label $y = 1$ is assigned to extensible inputs and $y = 0$ otherwise (Figure 2.1).

We will refer to the following running example throughout this chapter.

Example 1. Let $G = (V, E)$ be the graph with $V = \{0, 1, 2, 3\}$ and $E = \{\{0, 1\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$, and let $f : V_1 \to V_2$ be the partial automorphism with $V_1 = V_2 = \{1, 2, 3\}$ given by $f(1) = 2$, $f(2) = 1$, $f(3) = 3$ (the same instance shown in Figure 1.4 of Subsection 1.1.2). As established there, f is not extensible, so this instance carries label $y = 0$.

The core difficulty is that extensibility is a global property of G: it depends on the full orbit structure of $\mathrm{Aut}(G)$, which cannot be read off from bounded local neighborhoods. A partial map can be locally valid (preserving all adjacency constraints among mapped vertices) yet still non-extensible, because the vertices involved lie in the wrong orbits. This is precisely the gap that pseudo-similar vertices exploit: they are locally indistinguishable yet globally orbit-separated, so non-extensibility leaves no trace at any fixed neighborhood depth.

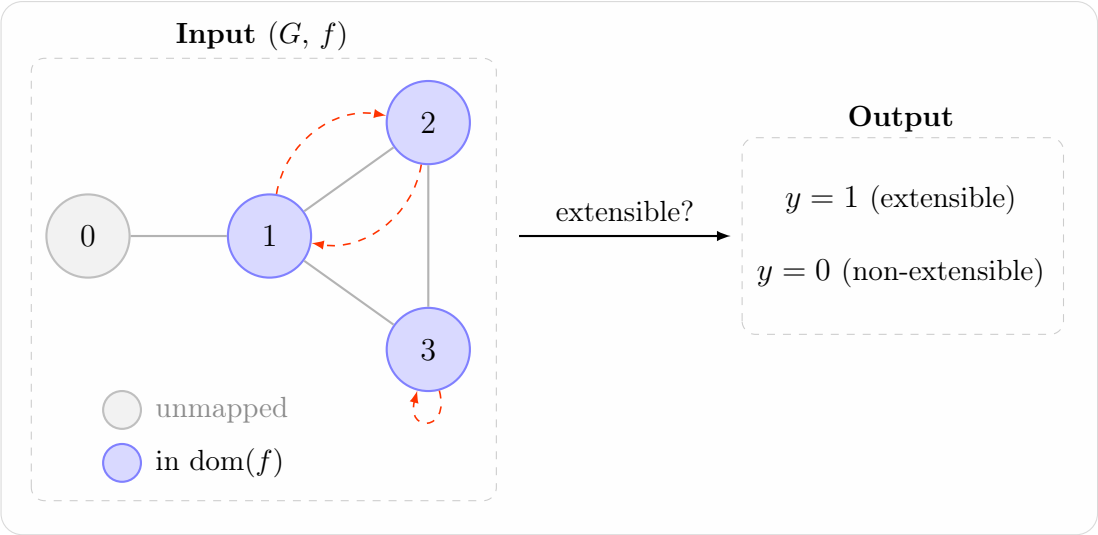


Figure 2.1: The partial automorphism extension problem as binary classification, illustrated using the running example (Example 1). Vertices in the domain of f are shaded blue; dashed arrows show the mapping. The model predicts whether f can be extended to a full automorphism of G.

2.2 Dataset construction

We collected 4,985 graphs from the House of Graphs database [9]. These graphs have 8–22 vertices and have $|\mathrm{Aut}(G)| \geq 2$, so that at least one non-trivial automorphism exists. As an upper-bound filter, we adopt the heuristic $|\mathrm{Aut}(G)| \leq \binom{n}{5}$ (where $n = |V(G)|$): we found empirically that this polynomial bound removes highly symmetric cases (complete, empty, star) without over-restricting the dataset.

The range of 8–22 vertices was selected based on domain considerations; the mean vertex count is ≈ 14.67 (Figure 2.2). The lower bound of 8 excludes graphs whose structure makes automorphism reasoning relatively straightforward.

2.2.1 Positive examples

For positive examples (i.e., partial automorphisms that can be extended), we used a simple strategy: for each graph G we computed its full automorphism group using McKay's `nauty` package [28], accessed through the `pynauty` library, then randomly selected a valid automorphism and restricted it to a random subset $V_1 \subseteq V$ containing between 50% and 80% of the graph's vertices. This strategy was motivated by the fact that every extensible partial automorphism is the restriction of some valid full automorphism of a given graph.

The 50–80% range was chosen to avoid degenerate extremes: below 50%, the domain is too sparse for the model to find meaningful evidence, since very small partial maps impose too few constraints to distinguish extensible from non-extensible; above 80%,

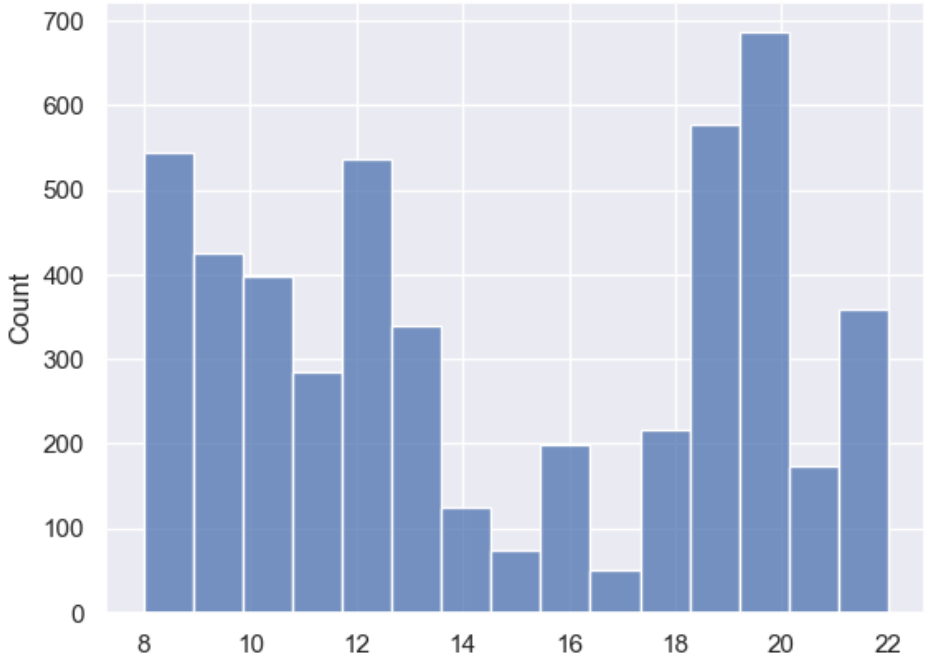


Figure 2.2: Distribution of vertex counts in the dataset.

the mapping nearly determines its own extension, collapsing the task to verifying the last few vertices rather than genuinely reasoning about extensibility. The range also matches the domain-size window used for negative examples (Subsection 2.2.2), so that $|\mathrm{dom}(f)|/|V(G)|$ does not leak the label.

2.2.2 Negative examples

Construction of negative examples required a more sophisticated strategy. We used two complementary methods: the pseudo-similar strategy and the blocking strategy. Although the pseudo-similar strategy is preferred when applicable, the Godsil–Kocay construction often yields no valid result, so the blocking strategy supplies the majority of negatives in practice.

Pseudo-similar strategy. This strategy builds on the notion of pseudo-similar vertices: two vertices $u, v \in V(G)$ with $G - u \cong G - v$ that nevertheless lie in distinct orbits of $\mathrm{Aut}(G)$. The seed mapping $\{u \mapsto v\}$ is a valid partial automorphism, since a single-vertex mapping trivially preserves adjacency, yet is non-extensible by construction: any full automorphism extending it would have to satisfy $\alpha(u) = v$, contradicting the orbit separation. Rather than searching for pseudo-similar vertices in an existing graph, we use the Godsil–Kocay construction to manufacture graphs containing such vertices by design. The strategy proceeds in two steps.

Step 1: Godsil–Kocay construction. For each base graph F from the dataset, a random non-identity automorphism $\sigma \in \mathrm{Aut}(F)$ is selected together with a random subset $S \subseteq V(F)$ satisfying $\sigma(S) \neq S$. Two new vertices u and v are appended to F: u is made adjacent to every vertex in S, and v is made adjacent to every vertex in $\sigma(S)$, yielding a graph G with $|V(F)| + 2$ vertices. By construction, the map τ that agrees with σ on $V(F)$ and sends u to v is an isomorphism $G - v \to G - u$, so u and v are pseudo-similar. The construction is accepted only when u and v lie in distinct orbits of $\mathrm{Aut}(G)$. Rather than running `nauty` on the larger graph G, we check a sufficient condition directly on F: if some $\varphi \in \mathrm{Aut}(F)$ satisfies both $\varphi(S) = \sigma(S)$ and $\varphi(\sigma(S)) = S$, then φ extends to an automorphism of G swapping u and v, placing them in the same orbit. The pair (σ, S) is rejected when such a φ exists. This check is sufficient to detect same-orbit pairs in the identified case, but not necessary: it may miss other same-orbit configurations, so some accepted candidates may still have u and v in the same orbit of $\mathrm{Aut}(G)$. Its purpose is to prune obvious failures cheaply, avoiding a `nauty` call on the larger graph G for every candidate; the quality control step (below) catches any same-orbit pairs that slip through by verifying non-extensibility directly. Note that G may have a trivial automorphism group even though every base graph F satisfies $|\mathrm{Aut}(F)| \geq 2$; the lower-bound filter applies only to the downloaded base graphs, not to the constructed ones. If no valid construction is found within a fixed number of attempts, the graph contributes no pseudo-similar negatives and the shortfall is absorbed by the blocking strategy.

Step 2: BFS expansion. Starting from the seed mapping $\varphi_0 = \{u \mapsto v\}$, the partial map is grown toward a target size t with $\lceil 0.5n \rceil \leq t \leq \lfloor 0.8n \rfloor$, where $n = |V(G)|$, via a breadth-first traversal of the domain, as described in Algorithm 1. The expansion maintains a FIFO queue Q initialized with u. At each step, the neighbors of the dequeued vertex are shuffled uniformly at random; for each unvisited neighbor x in that order, the algorithm asks the witness isomorphism for a partner: $y = \tau(x)$. The pair is skipped immediately if τ provides no image ($y = \perp$), meaning x falls outside the domain of τ, or if y is already paired with another vertex, which would violate injectivity. Only when y is available does the algorithm check whether adding $x \mapsto y$ preserves the automorphism property. Crucially, x is enqueued for further exploration only if the assignment succeeds; vertices for which no valid image exists are marked visited but not enqueued, so the BFS frontier expands exclusively through the mapped subgraph.

The subroutine ValidExtension(φ, x, y) checks whether augmenting φ with $x \mapsto y$ preserves the partial automorphism property. It validates the augmented mapping

Algorithm 1 BFS expansion using the witness isomorphism

Require: Graph $G = (V, E)$, seed vertices u, v, witness isomorphism $\tau\colon V(G{-}v) \to V(G{-}u)$, target size t

Ensure: Partial automorphism φ with $\varphi(u) = v$ and $|\varphi| \le t$, aiming for $|\varphi| = t$

$\varphi \leftarrow \{u \mapsto v\}$
$Q \leftarrow \textsc{Queue}(\{u\})$
visited $\leftarrow \{u\}$; used $\leftarrow \{v\}$
while $Q \neq \emptyset$ **and** $|\mathrm{dom}(\varphi)| < t$ **do**
    $w \leftarrow Q.\textsc{Dequeue}()$
    $\textsc{Shuffle}(N(w))$
    for each $x \in N(w)$ with $x \notin$ visited **do**
        visited $\leftarrow$ visited $\cup \{x\}$
        $y \leftarrow \tau(x)$ ▷ suggested partner for x
        if $y = \perp$ **then**
            continue ▷ τ undefined here; no partner available
        end if
        if $y \in$ used **then**
            continue ▷ partner already taken by another vertex
        end if
        if $\textsc{ValidExtension}(\varphi, x, y)$ **then**
            $\varphi(x) \leftarrow y$; used $\leftarrow$ used $\cup \{y\}$
            $Q.\textsc{Enqueue}(x)$
        end if
        if $|\mathrm{dom}(\varphi)| \ge t$ **then**
            break
        end if
    end for
end while
return φ

$\varphi \cup \{x \mapsto y\}$ by verifying, for every pair (z_1, z_2) in its domain, that

$$\{z_1, z_2\} \in E(G) \iff \{\varphi(z_1), \varphi(z_2)\} \in E(G), \tag{2.1}$$

and that the mapping is injective. Since all pre-existing pairs were valid before the extension, only pairs involving x need to be checked; the implementation checks all pairs for simplicity, giving $O(|\mathrm{dom}(\varphi)|^2)$ cost per BFS step rather than the $O(|\mathrm{dom}(\varphi)|)$ that an incremental check would achieve. If the BFS halts with $|\mathrm{dom}(\varphi)| < \lceil 0.5n \rceil$ (for instance, when the seed neighborhood admits too few valid images), the candidate is discarded and a fresh expansion is attempted up to a fixed budget.

Non-extensibility guarantee. A central property of the pseudo-similar strategy is that non-extensibility is *structurally guaranteed* by the seed and need not be checked after expansion.

Proposition 1. Let G be a graph and let $\varphi \supseteq \{u \mapsto v\}$ be any partial automorphism of G with u and v in distinct orbits of $\mathrm{Aut}(G)$. Then φ is non-extensible.

Proof. Suppose for contradiction that some $\alpha \in \mathrm{Aut}(G)$ extends φ. Then $\alpha(u) = \varphi(u) = v$, which would imply that u and v lie in the same orbit of $\mathrm{Aut}(G)$, contradicting the assumption. □

By Proposition 1, every output of the BFS expansion is non-extensible regardless of which vertices are added or in what order.

Blocking strategy. In the blocking strategy, for each base graph F, a random non-identity automorphism is selected from $\mathrm{Aut}(F)$ and restricted to a random subset $V_1 \subseteq V(F)$ of size t, where $\lceil 0.5n \rceil - 1 \leq t \leq \lfloor 0.8n \rfloor - 1$ with $n = |V(F)|$, giving an extensible partial automorphism φ. The blocking step then extends φ by a single cross-orbit assignment: an unmapped vertex x is paired with a target y from a different orbit of $\mathrm{Aut}(F)$, chosen such that the augmented mapping is still a valid partial automorphism. For tractability, the search bounds the number of candidate (x, y) pairs probed per blocking step to a small constant (five unmapped vertices and five cross-orbit targets per vertex in our implementation). By Proposition 1, the result is immediately non-extensible.

To increase difficulty, this blocking step is repeated up to k times, where k is drawn uniformly from $\{1, \ldots, \lfloor 0.8n \rfloor - t\}$. If a blocking step finds no valid cross-orbit pair within the candidate bound, iteration stops early; the mapping accumulated so far is used as the example.

Budget allocation. For each base graph F, letting $m_F = \min(m, |\mathrm{Aut}(F)|)$, we generate up to m_F positive examples on F, where m is a per-dataset cap (set to 10 for the baseline and extended-features datasets, and to 20 for the larger variant; see Section 2.2.3). When a Godsil–Kocay construction succeeds, the resulting graph G contributes up to $\lfloor m_F/2 \rfloor$ pseudo-similar negatives, together with an equal number of additional positive examples generated on G itself, so the constructed graph contributes balanced positives and negatives. The blocking strategy on F then fills the remaining negative budget so that the per-graph positive and negative counts stay matched: concretely, the number of blocking negatives requested is

$$|\mathrm{positives}_F| + |\mathrm{positives}_G| - |\mathrm{negatives}_{\text{pseudo-similar}}|.$$

When the pseudo-similar strategy cannot fill its quota, the shortfall is absorbed entirely by the blocking strategy.

Quality control. All generated examples are verified before inclusion in the dataset. Positive examples are guaranteed extensible by construction, since they are obtained as restrictions of known full automorphisms. Negative examples are verified to be (i) locally valid, preserving adjacency between all mapped vertices, and (ii) globally non-extensible. For both strategies condition (ii) follows from Proposition 1: pseudo-similar seeds are cross-orbit by construction, and each blocking step introduces a cross-orbit assignment that persists through all further iterations. The explicit non-extensibility check in the implementation therefore serves only as a sanity assertion. Examples failing either check are discarded.

Finally, the train/validation/test split is performed at the level of base graphs: no base graph F appears in more than one split. We additionally guard against constructed graphs G leaking across splits: each constructed G is identified by its `nauty` canonical certificate, and any G whose certificate has already been claimed by a different split is rejected. Validation and test examples are generated first, so train never reuses a constructed graph that is already in val or test. Together these checks prevent the model from exploiting graph-specific patterns and ensure that observed validation performance reflects genuine generalization to unseen graph structures.

2.2.3 Dataset statistics

Table 2.1 summarizes the statistics of the dataset, including the split sizes, positive and negative counts, their ratio, and the ratio of blocking to pseudo-similar examples within the negative class (Neg. B:PS). The relative size of a partial automorphism, measured as $|\mathrm{dom}(f)|/|V(G)|$, has a mean of 0.629 and a standard deviation of 0.090 across all examples. The per-class means differ slightly: positive examples have a

mean relative size of 0.644, while negative examples have a mean of 0.613. This gap is driven by the blocking strategy, whose negatives have a mean relative size of 0.604; pseudo-similar negatives have a mean of 0.640, closely matching the positive class. The difference arises because the blocking procedure initializes the domain one vertex below the positive sampling range and early termination of the cross-orbit extension loop is common in practice, leaving many blocking examples near the lower end of the size range. This weak correlation between domain coverage and label is a known limitation of the construction; The feature importance analysis (Table 3.5) confirms that the GIN does not exploit it as a shortcut, with error rate essentially flat across relative-size bins (Spearman $r = 0.007$, $p = 0.57$), so the $\approx 80\%$ accuracy ceiling is driven by structural difficulty rather than this confound.

Table 2.1: Dataset statistics. †Larger shares its validation and test sets with the Baseline & Extended Features datasets (rows above). *Neg. B:PS: ratio of blocking to pseudo-similar examples within the negative class.

Dataset	**Split**	**Total**	**Positive**	**Negative**	**Ratio**	**Neg. B:PS***
Baseline & Extended Features	Training	53385	27762	25623	1.08:1	3.26:1
	Validation	6732	3497	3235	1.08:1	2.98:1
	Test	6670	3429	3241	1.06:1	3.60:1
Larger†	Training	82184	43040	39144	1.10:1	2.88:1

The baseline and extended-features datasets share identical examples, differing only in the additional node features described below. Similarly, the baseline and larger datasets share the same validation and test sets, with the training set size being the only distinction; the larger training set is used in Chapter 3 to test whether additional training data improves generalization.

The roughly balanced positive-to-negative ratio is a deliberate diagnostic choice: the goal is to assess whether the model can learn to discriminate between extensible and non-extensible partial automorphisms given equal exposure to both classes. In practice, almost all partial automorphisms are non-extensible, so accuracy on this balanced test set measures discriminative ability rather than real-world predictive utility; a classifier that always predicts non-extensible would achieve only $\approx 50\%$ accuracy here.

Finally, we note that House of Graphs is a curated collection skewed toward graphs deemed interesting by researchers, so generalization to other graph families (e.g., random graphs or highly regular structures outside the automorphism filter) is not guaranteed and remains a direction for future work.

2.3 Vertex feature representations

A key question in our investigation concerns the utility of different vertex-feature representations. We evaluate two distinct feature sets to understand what information a GNN requires for effective automorphism-extension prediction.

Our baseline representation uses three features per vertex, all directly related to the partial automorphism. The first feature encodes vertex identity through a normalized vertex index node_id/$|V(G)|$.

The second and third features explicitly denote a vertex's role in the partial automorphism. For each vertex v, we record the mapping target as target_id/$|V(G)|$ if $v \in \mathrm{dom}(f)$ and the vertex maps to some target, or -1 if the vertex is not in the domain of the partial automorphism. Similarly, we record the mapping source as source_id/$|V(G)|$ if $v \in \mathrm{img}(f)$ and some source vertex maps to it, or -1 otherwise.

We remark that the node_id, target_id, and source_id are not permutation-equivariant: they depend on the integer labels assigned to vertices, so relabeling the graph changes the features. For target_id and source_id this is necessary: without some notion of vertex identity, the model has no way to determine which domain vertex maps to which image vertex, and normalized indices are the most direct encoding of the mapping within a standard node-feature GNN.

The node_id feature goes further: it exposes the raw vertex index, which carries no intrinsic structural meaning (the vertex labels are those assigned by the House of Graphs database and are not canonicalized). Relabeling the graph would change the node_id values and therefore the predictions, making the model sensitive to vertex ordering.

In practice this is not a limitation for the experiments reported here, since the labeling is consistent throughout the dataset, but it means the model is not invariant to vertex relabeling.

Formally, the baseline feature vector $\mathbf{x}_v \in \mathbb{R}^3$ is

$$\mathbf{x}_v = \left[\frac{\mathrm{id}(v)}{|V(G)|}, \frac{\mathrm{target}(v)}{|V(G)|}, \frac{\mathrm{source}(v)}{|V(G)|} \right], \tag{2.2}$$

where $\mathrm{target}(v) = f(v)$ if $v \in \mathrm{dom}(f)$ and -1 otherwise, and $\mathrm{source}(v) = f^{-1}(v)$ if $v \in \mathrm{img}(f)$ and -1 otherwise.

The use of -1 for unmapped vertices serves as a clear sentinel value, easily distinguishable from valid normalized vertex indices that fall in the range $[0, 1]$.

Table 2.2 shows the resulting feature values for each vertex of Example 1.

To test whether explicit structural features improve performance, we extended the baseline with four additional graph-theoretic properties, resulting in a seven-feature representation: vertex degree (normalized by the maximum degree in the graph), clustering coefficient, triangle count (normalized by the maximum triangle count in the

Table 2.2: Feature values for the example partial automorphism.

Vertex	node_id/$\lvert V(G)\rvert$	target_id/$\lvert V(G)\rvert$	source_id/$\lvert V(G)\rvert$
0	0/4	-1	-1
1	1/4	2/4	2/4
2	2/4	1/4	1/4
3	3/4	3/4	3/4

graph), and average neighbor degree (also normalized by the maximum average neighbor degree in the graph). The extended feature vector $\mathbf{x}'_v \in \mathbb{R}^7$ concatenates the baseline vector with these structural features:

$$\mathbf{x}'_v = \big[\, \mathbf{x}_v \parallel \mathbf{s}_v \,\big], \qquad \mathbf{s}_v = \big[\, \deg_{\text{norm}}(v),\ \text{coeff}(v),\ \text{tri}_{\text{norm}}(v),\ \text{neigh\_deg}_{\text{norm}}(v) \,\big]. \tag{2.3}$$

These structural features were computed using NetworkX, a Python library for graph processing [18]. By providing these features explicitly, we hypothesized that the network would learn automorphism extensibility more easily. Together, the problem formulation, dataset, and feature representations defined in this chapter provide the complete experimental setup for Chapters 3 and 4; we test the feature hypothesis first in Chapter 3.

Chapter 3

Baseline: GIN

In this chapter we evaluate the Graph Isomorphism Network (GIN), introduced in Subsection 1.3.2, on the partial automorphism extension task formulated in Chapter 2. GIN is the most expressive of the standard message-passing neural networks (MPNNs), a family of GNNs in which each vertex iteratively updates its embedding by aggregating messages from its neighbors (Chapter 1), and serves as the natural baseline: any task that GIN cannot solve is, at least with respect to 1-WL distinguishability (the 1-Weisfeiler–Leman color-refinement procedure that bounds what any standard MPNN can tell apart), out of reach for any standard MPNN.

We describe the model and training setup, present two experiments (feature engineering and dataset scaling), and analyze where the model succeeds and where it fails. The analysis shows that GIN's failures concentrate on globally non-extensible instances lacking a simple local witness, a regime that falls outside the expressive reach of 1-WL message passing. The failure modes identified here motivate the move to GraphGPS in Chapter 4.

3.1 Model architecture

We use the GIN-0 variant introduced in Subsection 1.3.2, in which the learnable scalar $\epsilon^{(k)}$ in Equation (1.2) is fixed to zero. Xu et al. [42] show that GIN-0 matches the learnable variant on standard graph classification benchmarks.

Per-layer MLP. The MLP inside each `GINConv` layer is a two-linear feed-forward network. For layer k (with $k = 1, \dots, K$), let d_{in} denote the input width: $d_{\text{in}} = d_0$ (the number of node features) for the first layer and $d_{\text{in}} = d$ (the hidden dimension) for all subsequent layers. The MLP maps

$$\mathbf{z} \;\mapsto\; W_2\,\mathrm{ReLU}(\mathrm{BN}(W_1\mathbf{z} + b_1)) + b_2, \qquad W_1 \in \mathbb{R}^{d\times d_{\text{in}}},\; W_2 \in \mathbb{R}^{d\times d},$$

where BN denotes batch normalization, which normalizes each feature across the training mini-batch to zero mean and unit variance and then applies a learned per-feature scale and shift; this reduces sensitivity to weight initialization and stabilizes gradient flow. The output width equals d throughout all layers, so vertex embeddings maintain constant dimension as they pass through the network.

After the `GINConv`, a second batch normalization is applied to the output, followed by a ReLU activation and dropout. This two-stage normalization pattern, one inside the MLP and one on the `GINConv` output, stabilizes training when the network is stacked to several layers.

Graph-level readout. After K message-passing layers, sum pooling is applied at each layer to obtain a per-layer graph representation, and these representations are summed across layers:

$$\mathbf{h}_G^{(k)} = \sum_{v \in V} \mathbf{h}_v^{(k)}, \quad k = 1, \ldots, K, \tag{3.1}$$

$$\mathbf{h}_G = \sum_{k=1}^{K} \mathbf{h}_G^{(k)}. \tag{3.2}$$

This multi-layer aggregation differs from the concatenation-based readout described in Subsection 1.3.2: rather than concatenating the per-layer sums, our implementation adds them, keeping the embedding dimension equal to d regardless of the number of layers. Both variants ensure that features at every WL depth (one message-passing layer corresponds to one WL color-refinement iteration) contribute to the final graph representation [42], but summation is in principle less information-preserving than concatenation: the per-layer sums are no longer recoverable individually, so two graphs whose layer-wise representations differ but happen to share the same total collapse to the same readout.

Classifier head. The graph embedding $\mathbf{h}_G \in \mathbb{R}^d$ is passed through a two-layer MLP to produce the binary prediction:

$$\hat{y} = W_{\text{cls}}\, \text{ReLU}(W_{\text{lin}}\, \mathbf{h}_G + b_{\text{lin}}) + b_{\text{cls}}, \quad W_{\text{lin}},\, W_{\text{cls}}^{\top} \in \mathbb{R}^{d \times d},$$

with dropout applied between the two layers. The network outputs a raw logit (an unconstrained real number representing the log-odds of the positive class, before any probability mapping is applied); the sigmoid is applied implicitly by the Binary Cross-Entropy with Logits loss during training and by thresholding at zero during inference.

3.2 Experimental setup

Training procedure. The model is implemented using PyTorch Geometric [13]. Our training procedure uses the widely adopted Adam optimizer [24], specifically its AdamW variant [27]. Weight decay is a regularization technique that adds a penalty proportional to the squared magnitude of the parameters to the training objective, discouraging the model from assigning large weights and thereby reducing overfitting. AdamW applies this penalty directly to the parameters instead of mixing it into the gradient update, giving more reliable regularization than plain Adam. We use Binary Cross Entropy with logits loss, which combines Binary Cross Entropy with a sigmoid layer for numerical stability: applying the sigmoid and the logarithm in a single fused operation avoids the overflow and underflow that occur when the two are computed separately on large or very negative logits. We also incorporate a ReduceLROnPlateau learning-rate scheduler that reduces the learning rate by a factor of 0.5 when validation accuracy plateaus for more than three consecutive epochs. To prevent overfitting, we use early stopping with patience of 15 epochs, terminating training if validation accuracy does not improve within this window, over a maximum of 150 epochs. Training was run on Kaggle's GPU environment.

Hyperparameter search. Hyperparameter search is the process of finding the combination of settings (learning rate, model size, regularization strength, etc.) that yields the best validation performance. Rather than trying configurations at random or on a fixed grid, Optuna [1] runs a sequence of *trials*. After each trial it fits a probabilistic model over the results seen so far and uses it to propose the next configuration, concentrating the search in regions that look promising. Trials that are clearly underperforming can be stopped early (*pruning*), freeing compute for more promising configurations.

We ran Optuna with 100 trials and median pruning (5 startup trials, 10 warm-up steps). Each trial trained for up to 50 epochs with early stopping patience of 5; the tighter budget allows quick comparison of configurations rather than full convergence. The objective was to maximize validation accuracy. The search space was: hidden dimension ($\{64, 128, 256, 512\}$), number of GIN layers ($\{2, 3, 4, 5, 6\}$), batch size ($\{64, 128, 256\}$), dropout rate (uniform in $[0.0, 0.5]$), learning rate (log-uniform in $[10^{-4}, 10^{-2}]$), and weight-decay coefficient (log-uniform in $[10^{-6}, 10^{-3}]$).

The best configuration found was: `hidden_dim` $= 256$, `num_layers` $= 3$, `dropout` ≈ 0.014, `lr` $\approx 9.0 \times 10^{-4}$, `weight_decay` $\approx 4.6 \times 10^{-4}$, `batch_size` $= 256$.

3.3 Experiments

Our experimental investigation consisted of two main experiments.

In the first experiment, we evaluated feature engineering by conducting separate Optuna optimizations for the baseline three-feature node representation and the enhanced seven-feature representation (Section 2.3). We then trained final models using the best configurations and compared their validation performance.

The second experiment assessed the effects of dataset scaling by training models on two dataset sizes using the same model configuration. The smaller dataset contained 53,385 examples, while the larger dataset contained 82,184 training examples, representing a $\approx 54\%$ increase in size.

Both experiments were repeated five times with different random seeds (seeds $0, 1, 2, 3, 4$), and the validation results are reported as mean $\pm$ s.d. over these runs; the test set was then evaluated using the single checkpoint with the highest individual validation accuracy among the five seeds.

3.4 Results

Through comprehensive hyperparameter optimization, we show that simple mapping-aware node features achieve 78.73% accuracy on the balanced test set described in Chapter 2 (roughly equal numbers of extendable and non-extendable examples). All reported numbers are on this balanced diagnostic regime; a classifier that always predicts non-extensible would achieve only $\approx 50\%$ here, so the gap reflects genuine learning of structural patterns. Our experiments further show that adding explicit structural features provides no significant performance improvement over the baseline node feature set (Table 3.1); the structural reason is explained in Section 3.5.2.

Table 3.1: Comparison of baseline and enhanced feature representations with optimized hyperparameters.

Metric	Baseline (3)	Enhanced (7)
Val Accuracy	0.7981 ± 0.0004	0.7981 ± 0.0027 (+0.00%)
Val F1 Score	$\mathbf{0.8145 \pm 0.0008}$	0.8143 ± 0.0041 (−0.02%)
Test Accuracy	0.7873	**0.7937** (+0.64%)
Test F1 Score	0.8032	**0.8124** (+0.92%)

Scaling the training set yielded a noticeable improvement in test accuracy, although the gain remained modest (roughly 1.7 percentage points on test for a $\approx 54\%$ increase in training data), as shown in Table 3.2. Additional data therefore still helps, but the

rate at which it converts into accuracy is small enough that we do not expect data scaling alone to lift performance much beyond the level observed here.

These findings are consistent with approximately 80% being a practical ceiling for standard GIN architectures on this task. Two dataset sizes are not sufficient to rule out a pure data ceiling; the argument that the plateau reflects model capacity rather than data scarcity rests on the 1-WL expressiveness bound examined in Section 3.5. Further improvements will likely require architectural innovations beyond conventional message-passing neural networks.

Table 3.2: Comparison of the baseline and larger training sets.

Metric	Baseline (53k)	Larger (82k)
Val Accuracy	0.7981 ± 0.0004	$\mathbf{0.8053 \pm 0.0034}$ (+0.72%)
Val F1 Score	0.8145 ± 0.0008	$\mathbf{0.8222 \pm 0.0032}$ (+0.77%)
Test Accuracy	0.7873	**0.8046** (+1.73%)
Test F1 Score	0.8032	**0.8195** (+1.63%)

3.5 Analysis

To understand where the GIN baseline fails, we evaluate the best checkpoint from the larger (82k) training run with the baseline three-feature representation on the test set of 6 670 graphs and analyze the structure of the residual errors. For every test graph we additionally pre-compute four structural metadata features used throughout the analysis below. These features were chosen to test three natural hypotheses: that larger graphs are harder (motivating `num_nodes`), that more symmetric graphs are harder because 1-WL struggles to distinguish vertices in regular structures (motivating `aut_grp_order`), and that the coverage of the partial map affects difficulty (motivating `paut_size` and `paut_relative_size`).

- `num_nodes`, the number of vertices,
- `paut_size`, the number of vertices on which f is defined,
- `paut_relative_size` = `paut_size` / `num_nodes`,
- `aut_grp_order`, the order of the automorphism group, computed with `pynauty`.

Overall test-set performance (accuracy 0.8046, $F_1 = 0.8195$, from Table 3.2) serves as the baseline for the error analysis below.

The $\approx 80\%$ of correctly classified instances concentrate in the structurally unambiguous regime. Non-regular graphs, which constitute 82% of the test set but only 79%

of the errors, are handled more reliably: vertex degrees provide a discriminating signal that local aggregation can exploit directly. Within the extendable class, recall reaches 0.86, indicating that GIN readily identifies partial automorphisms that are locally consistent with an extension, that is, cases where the neighborhood structure around every mapped vertex is compatible with the proposed correspondence. The remaining $\approx 20\%$ of errors are analyzed in the subsections below.

3.5.1 Confusion matrix analysis

A confusion matrix partitions test predictions into four cells defined by the true label and the predicted label. In this task, class 1 (extensible) is the *positive* class and class 0 (non-extensible) is the *negative* class, giving four cases: a *true positive* (TP) is an extensible graph correctly predicted extensible; a *true negative* (TN) is a non-extensible graph correctly predicted non-extensible; a *false positive* (FP) is a non-extensible graph incorrectly predicted extensible; a *false negative* (FN) is an extensible graph incorrectly predicted non-extensible. The classification report computes precision, recall, and F_1 for each class by treating that class as positive in turn:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad F_1 = \frac{2\,\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Precision is the fraction of predictions for that class that are correct; recall is the fraction of all true members of that class that are retrieved.

The per-class breakdown of the GIN model is reported in Table 3.3; the corresponding heatmap is shown in Figure 3.1. Errors are slightly biased toward false positives: precision is 0.84 on the negative class and 0.78 on the positive class. Recall on positives (0.86) substantially exceeds recall on negatives (0.74), confirming that the model is more reliable at identifying extendable partial automorphisms than at rejecting non-extendable ones. This asymmetry is not explained by class imbalance: the training set is balanced (roughly 1:1 positive-to-negative), so the model's tendency to over-predict the positive class reflects a genuine learned bias rather than a skewed prior.

Table 3.3: GIN classification report on the test set.

Class	Precision	Recall	F_1	Support
0 (no extension)	0.84	0.74	0.79	3241
1 (extension)	0.78	0.86	0.82	3429

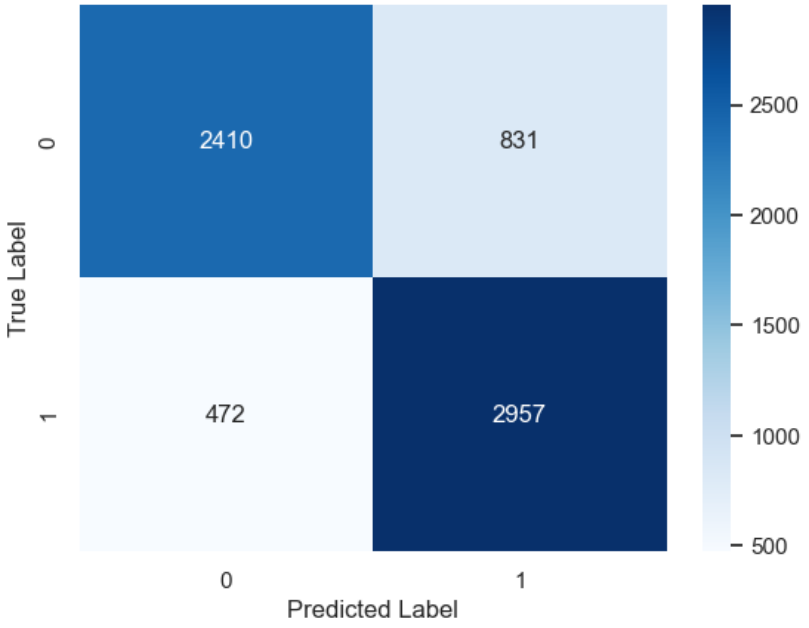


Figure 3.1: Confusion matrix of the GIN model on the test set.

3.5.2 Effect of graph regularity

A natural hypothesis is that GIN should fail more often on regular graphs, where degree information alone cannot distinguish vertices. Empirically the effect is mild: regular graphs make up 17.69% of the test set but 21.26% of GIN's errors. Throughout this analysis, chi-squared tests report χ^2 (how far observed counts deviate from what independence would predict), df (degrees of freedom, which parameterize the reference distribution), p (the probability of a deviation this large arising by chance), and $\varphi \in [0, 1]$ (effect size, where 0 means no association and 1 means perfectly predictive). A chi-squared test of independence confirms that the over-representation is statistically significant ($\chi^2 = 13.85$, df $= 1$, $p = 2 \times 10^{-4}$). However, the effect size is small: $\varphi = 0.046$, meaning that regular graphs are reliably more error-prone, yet the gap is too narrow for regularity alone to explain GIN's errors.

This finding offers a retroactive explanation for why the seven-feature representation fails to improve on validation over the three-feature baseline (Table 3.1). The four added features (normalized degree, clustering coefficient, triangle count, and average neighbor degree) carry little variance on regular graphs, where every vertex has the same degree and, in the most symmetric cases, the same local clustering structure. They thus collapse to nearly constant values precisely on the graphs where the model already struggles most, and provide no additional signal where it is needed.

3.5.3 Degree mismatch as a local witness

Any automorphism must map each vertex to a vertex of the same degree, so a pair $(v, f(v))$ with $\deg(v) \neq \deg(f(v))$ is an immediate witness that f cannot extend. This is a one-hop-local property: a model that aggregates each vertex's neighborhood and compares it to its image's neighborhood should, in principle, detect the obstruction in a single message-passing step.

To assess whether GIN exploits this signal, we check, for every non-extensible test graph, whether the partial automorphism contains at least one pair $(v, f(v))$ with $\deg(v) \neq \deg(f(v))$. Table 3.4 compares the rate of degree mismatches between GIN's false positives ($n = 831$) and true negatives ($n = 2{,}410$).

Table 3.4: Degree-mismatch rate among non-extensible test graphs, split by GIN prediction. Chi-squared test on the 2×2 contingency table: $\chi^2 = 72.99$, df $= 1$, $p = 1.3 \times 10^{-17}$, $\varphi = 0.150$.

Group	n	Degree-mismatch rate
All non-extensible	3241	58.50%
GIN true negatives (pred = 0)	2410	62.86%
GIN false positives (pred = 1)	831	45.85%

The result is the reverse of what a feature-integration failure hypothesis would predict. False positives have a *lower* rate of degree mismatches (45.85%) than true negatives (62.86%): GIN is better at rejecting non-extensible graphs when a degree mismatch provides a simple local witness. Computing the rejection rate within each subpopulation, GIN correctly rejects approximately 80% of degree-mismatch cases but only 67% of cases without a degree mismatch. The effect size $\varphi = 0.150$ is the largest observed across all analyses in this chapter.

GIN's primary failure mode is therefore non-extensibility *without* a simple local witness, precisely the regime requiring global structural reasoning that the 1-WL ceiling prevents local aggregation from supplying. The individual degree-mismatch failures discussed in Section 4.7 (Samples 639 and 3526) are real but atypical: they belong to the $\approx 20\%$ of degree-mismatch cases where GIN still fails despite the local witness being present.

3.5.4 Feature importance analysis

To understand which of the four structural features above predict GIN's per-graph errors, we use four complementary diagnostics. A *Random Forest classifier* gives feature importance scores, but these are known to be biased toward high-cardinality features;

we therefore also compute *permutation importance* on the same forest, which corrects for that bias by measuring how much accuracy drops when a feature's column is randomly shuffled. A *logistic regression* provides linear coefficients that reflect each feature's marginal contribution once the others are held fixed. Finally, *Spearman rank correlation* measures the monotonic association between each feature and the error indicator without fitting any model. These four diagnostics often disagree by design; the disagreements are themselves informative. Results from all four are summarized in Table 3.5.

Table 3.5: Structural-feature importance for predicting GIN errors.

Feature	RF Imp.	Perm. Imp.	LR Coef.	Spearman r	p
`aut_grp_order`	0.6288	0.0343	0.0000	+0.060	< 0.0001
`paut_relative_size`	0.1965	0.0115	1.5377	+0.007	0.5681
`num_nodes`	0.0943	0.0098	0.0558	−0.053	< 0.0001
`paut_size`	0.0804	0.0070	−0.1346	−0.048	0.0001

The logistic regression is fit with scikit-learn's defaults: L2 regularization (a penalty added to the training objective that discourages large coefficients by shrinking them toward zero, controlled by an inverse-strength parameter C, here $C = 1.0$) and no feature standardization (i.e., features are used as-is, without rescaling them to comparable ranges). The table reports raw coefficients rounded to four decimals, so the entry 0.0000 for `aut_grp_order` is a rounded value ($\approx -10^{-4}$) rather than an exact zero. Because the features differ in natural scale (`aut_grp_order` reaches into the tens of thousands while the others are at most single-digit), coefficient magnitudes are not directly comparable across rows; the comparisons below should be read together with the permutation importance and Spearman columns.

Three observations stand out.

First, the gap between RF importance (0.63) and permutation importance (0.034) for `aut_grp_order` is a symptom of Random Forest's cardinality bias: `aut_grp_order` takes many distinct values, so the tree has many candidate thresholds to split on and tends to use this feature often, inflating its importance score even when those splits carry little predictive signal. Permutation importance is the more trustworthy diagnostic in this setting, and demotes `aut_grp_order` accordingly.

Second, `paut_relative_size` receives by far the largest logistic regression coefficient; once this relative measure is in the model, the absolute group size adds essentially nothing. This may appear to conflict with `paut_relative_size`'s near-zero Spearman correlation ($r = 0.007$, $p = 0.57$): Spearman measures the marginal relationship between a single feature and the error indicator in isolation, whereas the logistic regression

coefficient reflects each feature's contribution after the others are held fixed. The relative size thus adds predictive signal once the absolute counts are controlled for, even though it carries little information on its own.

Third, every Spearman effect size is small ($|r| < 0.07$), which means that none of these summary features individually predicts errors with practical strength.

The honest conclusion is that the four structural features in aggregate do not strongly explain where GIN fails. Either the failure mode lies in feature interactions not captured here, or hard cases are hard for reasons that no single summary statistic can encode.

3.6 Synthesis

Within the GIN model in isolation, no single structural feature predicts errors with meaningful effect size. The asymmetric error profile (precision 0.84 on negatives versus 0.78 on positives) and the over-representation of regular graphs among errors together point to a symmetry-driven failure mode: the model's mistakes concentrate on locally coherent but globally non-extensible mappings, the kind that the pseudo-similar negative-generation strategy in Section 2.2.2 was designed to produce (negatives built from vertices u, v with $G - u \cong G - v$ yet lying in distinct orbits, so the mapping looks locally valid but cannot be extended to any full automorphism). Resolving these cases requires reasoning that conventional message passing cannot easily express: the message-passing framework is bounded by the expressive power of the 1-WL test [42], and automorphism extension requires distinguishing graph structures that 1-WL cannot separate, which explains the observed accuracy ceiling.

This motivates the architectural change pursued in Chapter 4, where local message passing is supplemented with global attention and positional encodings designed to break precisely the symmetries that defeat the baseline.

Chapter 4

Beyond 1-WL: GraphGPS

The results of Chapter 3 pointed to a single dominant failure mode of standard GIN on the partial automorphism extension task: locally coherent but globally non-extensible mappings, where resolving extensibility requires reasoning about global structure that cannot be recovered from iterated local aggregation. Further analysis reveals no single structural summary feature that predicts errors with meaningful strength.

This failure mode stems from the 1-WL ceiling (Subsection 1.3.3). To address it, this chapter studies GraphGPS [33], a hybrid architecture that combines message passing with global self-attention and adds explicit positional and structural encodings. The positional encodings break the 1-WL symmetry, while the attention mechanism provides the global information flow that pure MPNNs lack. Although GraphGPS reduces the error rate 8.7x, residual errors concentrate strongly on regular graphs. The architecture's advantage is empirically localized to instances where the partial automorphism maps vertices across long relative distances.

4.1 Graph transformers

The Transformer architecture [37] replaced recurrent and convolutional models in natural language processing by relying entirely on self-attention. Each token in a sequence attends to every other token in a single layer, giving the model an effectively unbounded receptive field at constant depth. Adapting this idea to graphs is attractive precisely because pure message passing has the opposite property: information propagates only one hop per layer, which is the root cause of the failure mode observed in Chapter 3.

A naive adaptation, however, does not work. Self-attention treats its input as an unordered set, so a Transformer applied directly to the multiset of node feature vectors discards the edges entirely and becomes invariant to graph structure. Two non-isomorphic graphs with the same multiset of node features collapse to the same representation. The information that ordinary Transformers recover from positional

encodings on sequences (which token comes first, which comes last) has no graph analogue, since a graph has no canonical linear ordering.

Existing graph transformers all respond to this problem by reintroducing graph structure into the attention block, typically through one or both of two mechanisms: positional and structural encodings appended to node features, and attention biases that depend on shortest-path distance, edge type, or neighborhood. The architectures differ in which mechanism they emphasize and in whether they retain any local message-passing component.

Two broad families have emerged. *Pure* graph transformers replace message passing entirely with global self-attention and rely on encodings and biases for graph structure. The Graph Transformer of Dwivedi and Bresson [11] restricts attention to graph neighbors and augments node features with Laplacian eigenvector positional encodings. Graphormer [44] keeps attention fully global but injects shortest-path distance, node-degree, and edge encodings as attention biases, and reached state-of-the-art results on several molecular benchmarks. *Hybrid* architectures, by contrast, combine local message passing with global self-attention inside each layer. GraphGPS [33], the subject of this chapter, is the prototypical example: it preserves the locality bias of MPNNs while letting the attention sub-layer transport information between distant nodes in a single hop. A comprehensive survey of both families is given by Shehzad et al. [35].

4.2 Positional and structural encodings

The failure mode of message passing identified in Chapter 3 ultimately comes back to a single fact: two nodes with 1-WL-equivalent neighborhoods receive identical embeddings, regardless of how many layers are stacked. The standard remedy is to give each node a feature that already distinguishes it from its 1-WL-equivalent peers before message passing begins. Positional and structural encodings (jointly abbreviated PE/SE) do exactly this [12]: they precompute a graph-aware fingerprint for each node and attach it as an additional input feature. Positional encodings locate a node within the global structure of the graph, while structural encodings summarize the local geometry around it.

Laplacian positional encodings. The core question positional encoding answers is: *where is this node in the global structure of the graph?* For sequences this is easy – a token at position i is simply assigned the index i. For graphs there is no canonical ordering, so a different approach is needed.

Let A be the adjacency matrix of the graph ($A_{uv} = 1$ if u and v are connected, 0 otherwise) and D the diagonal degree matrix ($D_{vv} = \deg(v)$). The normalized graph Laplacian $L = I - D^{-1/2}AD^{-1/2}$ is built from these two matrices: it equals the identity

minus a degree-normalized version of A. It can be diagonalized as $L = \sum_{i=1}^{n} \lambda_i \phi_i \phi_i^\top$, where $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues and each eigenvector $\phi_i \in \mathbb{R}^n$ assigns one real number to every node. Eigenvectors with small eigenvalues vary slowly across the graph, so nearby nodes receive similar values; eigenvectors with larger eigenvalues vary quickly and encode finer structural distinctions.

The k-dimensional Laplacian positional encoding (LapPE) of node v is formed by stacking the values that the k lowest non-constant eigenvectors assign to v [11, 12]:

$$\text{LapPE}(v) \;=\; \big(\phi_2(v),\ \phi_3(v),\ \ldots,\ \phi_{k+1}(v)\big).$$

The first eigenvector ϕ_1 is constant on every connected component and carries no positional information, so it is discarded. Crucially, two nodes that are 1-WL-equivalent need not share the same LapPE coordinates, so the encoding can break symmetries that message passing cannot.

Two practical issues affect LapPE. First, each eigenvector is defined only up to sign: ϕ_i and $-\phi_i$ are both valid eigenvectors, and when eigenvalues repeat the ambiguity extends to entire subspaces. We follow the GraphGPS recipe [33] and handle this by randomly flipping signs at each forward pass during training, so the model learns to be robust to the ambiguity. Second, only the first k eigenvectors are retained because computing the full spectrum is expensive ($O(n^3)$) and most positional information sits in the low-frequency end.

The choice we make. We use LapPE with $k = 5$ throughout this chapter. The task we solve, deciding extensibility of a partial automorphism, is global: it requires comparing distant vertices and reasoning about graph-wide symmetry. LapPE supplies exactly this kind of long-range positional information and explicitly breaks the 1-WL symmetry that capped GIN at roughly 80% accuracy.

We also experimented with random-walk structural encoding (RWSE) [12, 33], which assigns each node a vector of return probabilities: the probability that a random walk starting at v returns to v after $1, 2, \ldots, k$ steps. RWSE characterizes the neighborhood of a node and is useful for detecting short cycles and similar motifs, but it is invariant under graph automorphisms. Two nodes related by a non-trivial automorphism share identical RWSE vectors, so the encoding cannot distinguish the very pairs that our task hinges on. In practice RWSE produced results that were not far below LapPE, but LapPE's ability to assign distinct coordinates to symmetry-related nodes makes it the principled choice for this task.

We set $k = 5$, which lies within the safe range for our graph sizes (8–22 nodes). This captures the five most structurally informative eigenvectors of the normalized Laplacian, encoding global connectivity, rough symmetry, and spectral cluster structure, while ensuring that every graph in the dataset has enough eigenvectors available

(a graph with n nodes has only $n-1$ non-constant eigenvectors, so k must stay below that bound even for the smallest graphs). The choice balances representational richness against model complexity.

4.3 The GraphGPS architecture

GraphGPS [33] is presented by its authors as a *recipe* rather than a fixed model: it specifies how to compose a local message-passing sub-layer, a global self-attention sub-layer, and a positional or structural encoding into a layer that can be stacked, but the concrete choice of MPNN, of attention, and of encoding is left to the user. The chapter uses the most standard instantiation: GIN as the MPNN, dot-product multi-head self-attention as the global block, and Laplacian eigenvector positional encodings.

Layer recipe. Let n be the number of nodes, d the hidden dimension, $X^{(\ell)} \in \mathbb{R}^{n\times d}$ the node embeddings at the input of layer ℓ, and $E \subseteq V \times V$ the edge set. A single GPS layer produces $X^{(\ell+1)}$ by computing a local update and a global update in parallel and combining them:

$$\begin{aligned} Y_M^{(\ell)} &= \mathrm{MPNN}^{(\ell)}\big(X^{(\ell)},\, E\big), \\ Y_A^{(\ell)} &= \mathrm{Attn}^{(\ell)}\big(X^{(\ell)}\big), \\ X^{(\ell+1)} &= \mathrm{FFN}^{(\ell)}\big(Y_M^{(\ell)} + Y_A^{(\ell)}\big), \end{aligned}$$

with residual connections and batch normalization applied inside each sub-layer [33]. The sum $Y_M^{(\ell)} + Y_A^{(\ell)}$ merges the local and global updates before being passed to $\mathrm{FFN}^{(\ell)}$, a position-wise two-layer MLP that produces the final output embedding for layer ℓ.

The MPNN sub-layer is any standard message-passing block; following the original recipe we use GIN [42], which keeps the comparison with Chapter 3 controlled: the local aggregation is identical, and only the global attention path is new. The attention sub-layer is a vanilla multi-head self-attention block [37] over the multiset of node embeddings, with no mask: every node attends to every other node in the graph, irrespective of edges.

Where the encodings enter. The positional encoding is injected only at the input of the first layer. Following Section 4.4, for each node v the LapPE vector $p_v \in \mathbb{R}^k$ is passed through a learned linear projection and added to the projected node features,

$$X_v^{(0)} \;=\; W_x x_v + W_p p_v,$$

where $x_v \in \mathbb{R}^{d_{\text{in}}}$ is the raw node feature vector (Section 2.3) and $W_x \in \mathbb{R}^{d\times d_{\text{in}}}$, $W_p \in \mathbb{R}^{d\times k}$ are learned weight matrices. From that point onward the encoding lives only

inside the learned embedding; subsequent layers see $X^{(\ell)}$ and do not access p_v directly. This is the simplest of the strategies discussed in [33] and is sufficient for our purposes, since the attention sub-layer can propagate the encoding information across the whole graph in a single hop.

Expressivity. The expressivity gain over a pure MPNN is straightforward to state. A pure message-passing network is bounded by 1-WL (Subsection 1.3.3). A GPS layer combines the same MPNN with a global attention block over node features augmented by a positional encoding. The key condition is *injectivity of the PE on 1-WL-equivalent nodes*: the encoding must assign distinct vectors to every pair of nodes that 1-WL cannot tell apart. When this holds, the augmented input already separates those nodes before any message passing begins, and every subsequent layer preserves the distinction. The attention sub-layer then distributes this information globally, restoring the long-range reasoning that no fixed-depth MPNN can perform. Rampášek et al. [33] make this argument formal and show that the injectivity condition is sufficient for GraphGPS to be strictly more expressive than 1-WL.

For LapPE the sufficient condition for injectivity is that the normalized Laplacian has a *simple spectrum* (all eigenvalues distinct). When no eigenvalue repeats, each eigenvector is determined up to a global sign flip, and nodes in structurally distinct positions generally receive different eigenvector coordinates; the sign ambiguity is neutralized by the random-flip training procedure described above. The assumption fails, however, on *vertex-transitive* graphs: graphs where for any two vertices u and v there exists an automorphism mapping u to v (all k-regular graphs are an example). On such graphs, every node receives the same multiset of LapPE values, so the encoding cannot break the symmetry that 1-WL also cannot break. This is precisely why, in Section 4.6.2, the residual errors of GraphGPS concentrate on regular graphs: the injectivity condition fails there by construction, and no choice of k or training procedure can fix it.

4.4 Experimental setup

Dataset and splits. We use the larger (82k) training set from Chapter 3 with the same 80/10/10 train/validation/test split, so all results are directly comparable against the strongest GIN baseline (Table 3.2).

Node features and positional encodings. Each node is described by the same three-feature baseline representation introduced in Section 2.3: node identity, the target of the partial automorphism mapping from this node (if any), and the source that maps to this node (if any). We augment each graph with LapPE ($k = 5$; Section 4.2).

Training procedure. The training procedure follows Section 3.2 exactly: AdamW optimizer with Binary Cross Entropy with logits loss, a ReduceLROnPlateau scheduler (factor 0.5, patience 3), and early stopping with patience of 15 epochs over a maximum of 150 epochs.

Hyperparameter search. We ran Optuna with 100 trials and median pruning (10 startup trials, 8 warm-up steps). The search space matches the GIN search (Section 3.2) with one addition: the number of attention heads was fixed at 4 throughout to keep the search tractable. The remaining search space was: hidden dimension ($\{64, 128, 256, 512\}$), number of GPS layers (2–6), batch size ($\{64, 128, 256\}$), dropout (uniform in $[0.0, 0.5]$), learning rate (log-uniform in $[10^{-4}, 10^{-2}]$), and weight decay (log-uniform in $[10^{-6}, 10^{-3}]$).

The best configuration found was: `hidden_dim` $= 128$, `num_layers` $= 6$, `num_heads` $= 4$, `dropout` ≈ 0.010, `lr` $\approx 2.5 \times 10^{-4}$, `weight_decay` $\approx 3.4 \times 10^{-4}$, `batch_size` $= 64$.

Implementation. The model is implemented using the `GPSConv` layer from PyTorch Geometric [13], which internally combines a `GINConv` message-passing sub-layer with a multi-head self-attention sub-layer. LapPE is computed using PyTorch Geometric's `AddLaplacianEigenvectorPE` transform. Training was run on Kaggle's GPU environment.

4.5 Results

We evaluate GraphGPS on the same held-out test set of 6 670 graphs. Table 4.1 reports the results. GraphGPS is correct on roughly 98% of test graphs, compared to 80% for GIN.

Table 4.1: Test-set performance of GIN and GraphGPS. Accuracy is the fraction of correctly classified graphs. F_1 (pos.) is the F_1 score on the positive class (partial automorphisms that *can* be extended); it balances precision and recall and is reported separately because the two classes have slightly different sizes.

Model	Accuracy	F_1 (pos.)
GIN	0.8046	0.8195
GraphGPS	0.9777 (+17.31%)	0.9784 (+15.89%)

This gap is not explained by the dataset being too small or too noisy for GIN to learn from. Both models see the same training data; GraphGPS simply has a richer architecture that can express distinctions GIN cannot. The 80% ceiling of Chapter 3

is therefore a consequence of GIN's architectural limitations, not a property of the problem itself.

The remainder of this chapter examines where, specifically, GraphGPS succeeds and where it still falls short.

4.6 Analysis: revisiting GIN's failure modes

4.6.1 Confusion matrix

The GraphGPS classification report (Table 4.2; TP, TN, FP, FN, precision, recall, and F_1 are defined in Section 3.5.1) shows balanced and near-uniform performance across both classes, with precision and recall at 0.97–0.98 on both. The asymmetric error profile observed for GIN (precision 0.84 on the negative class versus 0.78 on the positive class, indicating an excess of false positives over false negatives) is essentially absent.

Table 4.2: GraphGPS classification report on the test set.

Class	Precision	Recall	F_1	Support
0 (no extension)	0.98	0.97	0.98	3241
1 (extension)	0.97	0.98	0.98	3429

4.6.2 Effect of graph regularity

Although GraphGPS reduces the absolute error count by roughly 89%, its remaining errors are disproportionately concentrated on regular graphs: 46.31% of GraphGPS errors occur on regular graphs versus only 17.69% in the test population. This is a far stronger over-representation than for GIN (21.26%). Once GraphGPS has eliminated the easier failure modes, what remains is a residual class of hard symmetric instances that even Laplacian positional encoding does not fully resolve.

To probe whether vertex-transitive graphs specifically drive the concentration (the LapPE injectivity argument of Section 4.3 predicts they should), Table 4.3 breaks the test set into three subgroups and reports GPS error rates and representation among errors for each.

Within regular graphs, a chi-squared test finds no significant difference between vertex-transitive and non-vertex-transitive instances ($\chi^2 = 0.74$, $p = 0.39$, $\varphi = 0.025$). Notably, the observed direction of the trend is the opposite of what the LapPE injectivity argument would predict: vertex-transitive graphs, the subgroup for which LapPE is provably unable to break symmetry, have a *lower* empirical error rate (4.12%) than

Table 4.3: GraphGPS error rates and share of total errors by regularity and vertex-transitivity. Vertex-transitive graphs are a strict subset of regular graphs (170 of 1,180 regular graphs, i.e. 14.4%).

Subgroup	n	Error rate	Share of errors
Non-regular	5490	1.46%	53.69%
Regular, not vertex-transitive	1010	6.14%	41.61%
Regular, vertex-transitive	170	4.12%	4.70%

non-vertex-transitive regular graphs (6.14%). The difference is not statistically significant, but its direction undermines the sharpest form of the injectivity explanation. The error concentration is instead carried by the larger non-vertex-transitive regular subgroup, which makes up 15.1% of the test population but 41.6% of errors. The deeper cause of the regularity concentration therefore remains open.

4.6.3 Feature importance analysis

The same four structural features used in Section 3.5.4 (namely `num_nodes`, `paut_size`, `paut_relative_size`, and `aut_grp_order`) are evaluated against the GraphGPS error indicator in Table 4.4. With far fewer errors to model, the absolute permutation importances are roughly an order of magnitude smaller than for GIN, and the same caveats about Random Forest cardinality bias apply.

Table 4.4: Structural-feature importance for predicting GraphGPS errors.

Feature	RF Imp.	Perm. Imp.	LR Coef.	Spearman r	p
`aut_grp_order`	0.6100	0.0035	0.0000	+0.058	< 0.0001
`paut_relative_size`	0.1590	0.0010	−0.2921	−0.054	< 0.0001
`num_nodes`	0.1157	0.0010	+0.1940	+0.045	0.0002
`paut_size`	0.1153	0.0008	−0.2152	+0.020	0.0962

4.6.4 Effect of relative partial automorphism size

Spearman correlation of `paut_relative_size` with the error indicator is $r = -0.054$ ($p < 0.0001$): GraphGPS makes slightly fewer mistakes when a larger fraction of the vertex set is constrained by the partial automorphism. The effect size is small. Figure 4.1 visualizes this relationship.

Notably, the logistic regression coefficient for `paut_relative_size` reverses sign between the two models: +1.54 for GIN (Table 3.5) versus −0.29 here. Under GIN,

larger relative coverage was associated with more errors once absolute counts were controlled for; under GraphGPS the direction inverts. This is consistent with the global-reasoning interpretation developed in Section 4.7.2: a partial automorphism constraining a larger fraction of the graph forces longer-range consistency checks, which GraphGPS's attention and positional encoding are better equipped to exploit than the 1-WL-bounded GIN, though the effect size is modest.

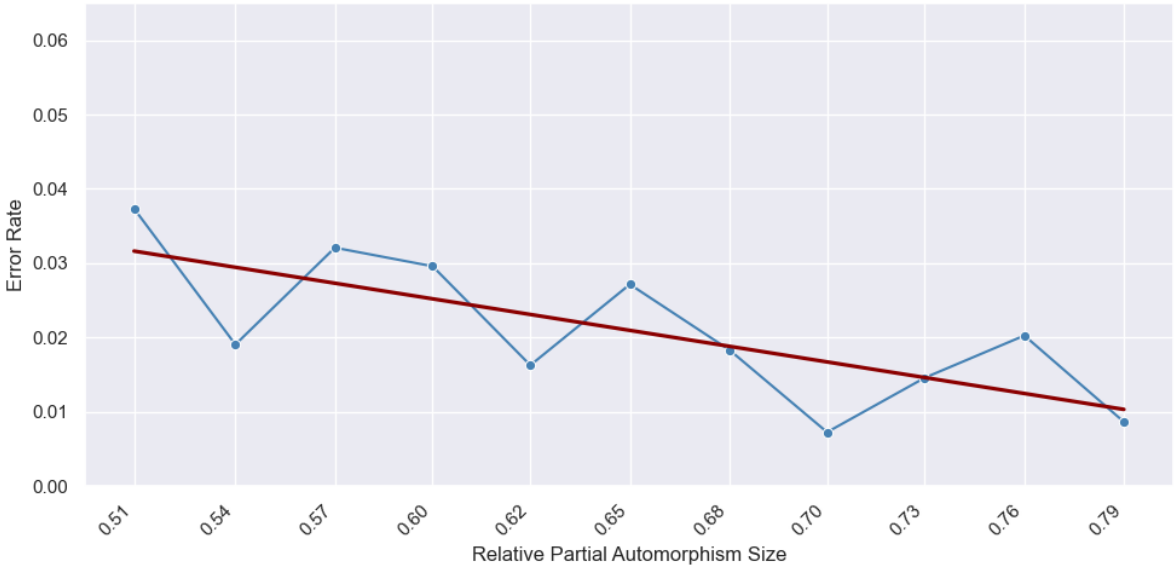


Figure 4.1: GraphGPS error rate as a function of `paut_relative_size`.

4.6.5 Effect of automorphism group order

Figure 4.2 compares GIN and GraphGPS error rates across automorphism group order buckets, binned on a logarithmic scale.

GIN's error rate is nearly flat across all non-trivial symmetry buckets (approximately 0.21 for group orders 2 through 4608), falling only for graphs with a trivial automorphism group (order 1, error 0.03). Once any non-trivial symmetry is present, GIN fails at roughly the same rate regardless of how large the symmetry group is. This is consistent with the 1-WL ceiling: the barrier is binary rather than graded.

GraphGPS reduces errors in every bucket, roughly nine times. The moderate symmetry range (group orders 17–277) retains the highest GPS error rate (≈ 0.04), while the most extreme bucket (orders > 1130) drops to 0.019. The pattern is non-monotone and the counts in the high-symmetry buckets are small ($n < 230$), so no strong claim about ordering can be made there. The central result is qualitative: no bucket is left behind, and GraphGPS does not specifically collapse on the most symmetric graphs.

The RF importance of `aut_grp_order` for predicting GraphGPS errors (0.61, Table 4.4) reflects cardinality bias rather than a genuine monotone relationship between

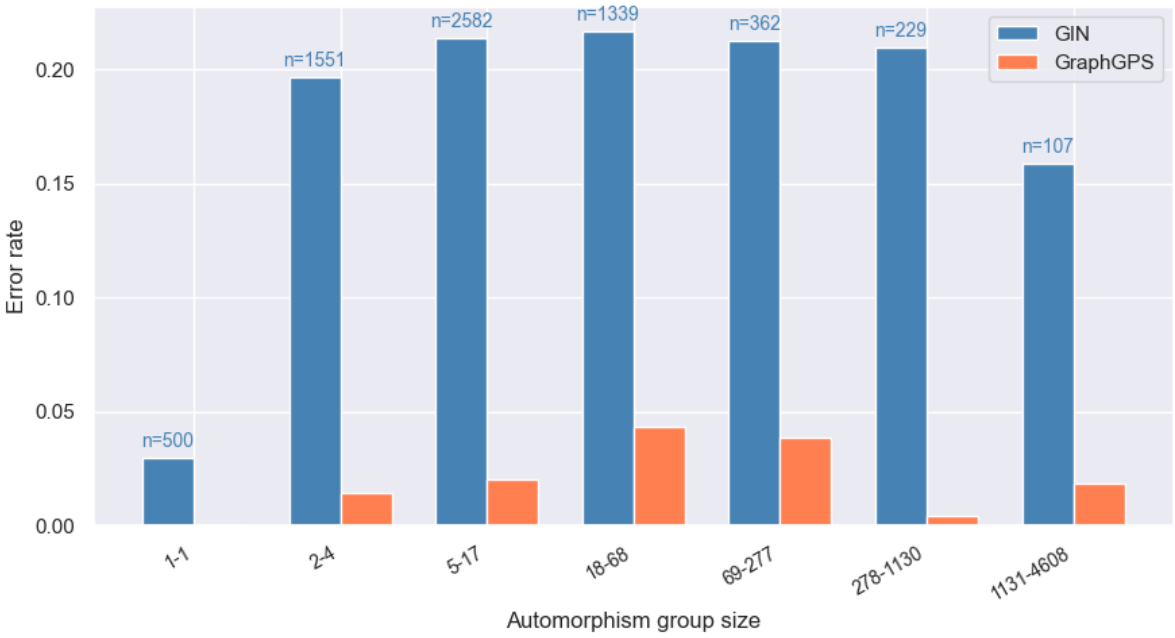


Figure 4.2: Error rate by automorphism group order bucket for GIN and GraphGPS. Bin counts (shown above GIN bars) are identical for both models.

group order and error rate. The Spearman correlation ($r = +0.058$) is the more honest measure, and it is small.

4.7 Comparative analysis: where does GraphGPS help?

The marginal analyses above show that no single structural feature strongly predicts errors in either model. We therefore turn to a paired comparison: for each test graph we have the GIN prediction and the GraphGPS prediction, and we ask what distinguishes the graphs on which GraphGPS is confidently right, but GIN is wrong from the graphs on which GIN is right.

4.7.1 Confident GIN failures corrected by GraphGPS

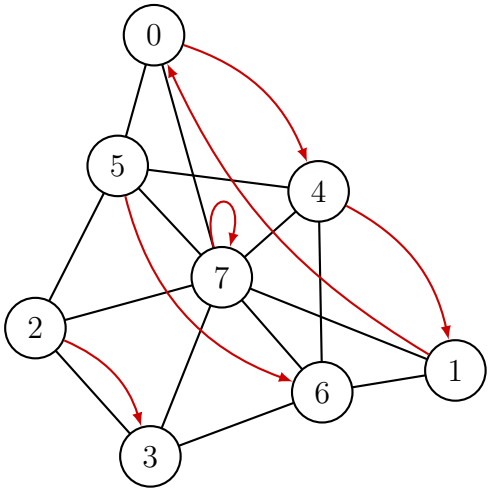
Restricting to test graphs of 8 vertices (the smallest size in the dataset), we identify 90 instances where GraphGPS is correct and GIN is wrong with non-trivial confidence ($|p - 0.5| > 0.3$, where $p \in [0, 1]$ is the model's predicted probability of extensibility; the decision boundary is $p = 0.5$, so this threshold keeps only predictions that are at least 0.3 away from the boundary on the wrong side). Three representative cases are shown in Figure 4.3. In all three, $|\hat{p}_{\text{GIN}} - 0.5| > 0.48$, meaning GIN is nearly maximally confident in the wrong answer, whereas GraphGPS assigns $\hat{p}$ indistinguishable from 0 or 1 to the correct class. Red arrows show f; self-loops mark fixed points.

Sample 639 and Sample 3526 (label: no extension). Both are non-extensible graphs that GIN confidently misclassifies as extensible. In each case the obstruction is a degree mismatch: any automorphism must map each vertex to a vertex of the same degree, yet in Sample 639 the mapping $f(0) = 4$ pairs a degree-2 vertex with a degree-3 vertex (and $f(4) = 1$ does the reverse), and in Sample 3526 the transposition $f(4) = 5$, $f(5) = 4$ pairs vertices of degrees 2 and 4. No full automorphism can extend such a mapping, regardless of what happens on the remaining vertices. Degree is a one-hop-local quantity that GIN can in principle compute, yet it still assigns near-certainty to the wrong class.

Sample 4060 (label: extensible). This graph has a bilateral symmetry: two identical wings $\{1, 2, 3\}$ and $\{4, 5, 6\}$, each forming a star (center 1 resp. 4) whose center connects to the other two wing vertices and to a common hub (node 7). The two wings are further linked by cross-wing edges 2–5 and 3–6, which are what close the bilateral symmetry. A pendant vertex 0 is attached only to the hub. The partial automorphism $f(0)=0$, $f(1)=4$, $f(2)=5$, $f(3)=6$ maps the left wing to the right wing and fixes the hub-adjacent vertices. It extends to the full automorphism ϕ that swaps the two wings entirely: $\phi(1)=4$, $\phi(2)=5$, $\phi(3)=6$, $\phi(4)=1$, $\phi(5)=2$, $\phi(6)=3$, with 0 and 7 fixed. Despite the symmetry being global and structurally explicit, GIN assigns probability 0.01 to extensibility.

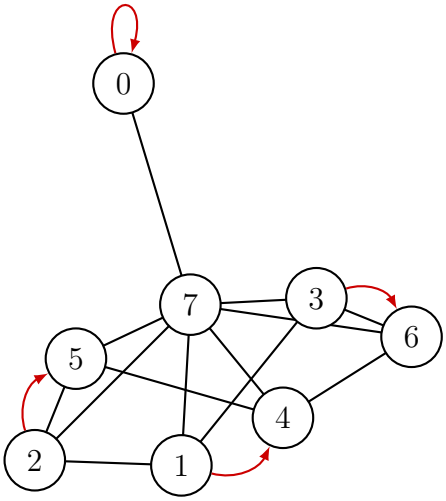
These cases illustrate two distinct regimes of GIN failure. In the first regime (Sample 639 and Sample 3526), the obstruction is locally detectable: degree is a one-hop quantity, so a model that correctly integrates the mapping features with neighborhood structure should catch the mismatch. That GIN assigns near-certainty to the wrong answer in these instances is a genuine failure of feature integration. However, as the population-level analysis in Section 3.5.3 shows, such cases are not GIN's primary failure mode: across the full test set, GIN correctly rejects approximately 80% of non-extensible graphs whose obstruction is a degree mismatch, performing better there than on instances without a simple local witness (67% rejection rate). Samples 639 and 3526 are atypical examples of this minority failure. In the second regime (Sample 4060), the relevant structure is the bilateral symmetry of the whole graph, which no fixed number of local aggregation steps can reliably identify. This is the more representative failure mode: non-extensibility without a local witness that GIN can detect. It is precisely the kind of failure that global attention is designed to address, and it motivates the structural hypothesis developed below.

The reverse direction is negligible: 73 samples (1.09% of the test set) are classified correctly by GIN but not by GraphGPS. Given GraphGPS's strictly greater expressive power, this is attributed to training variance rather than a systematic gap.



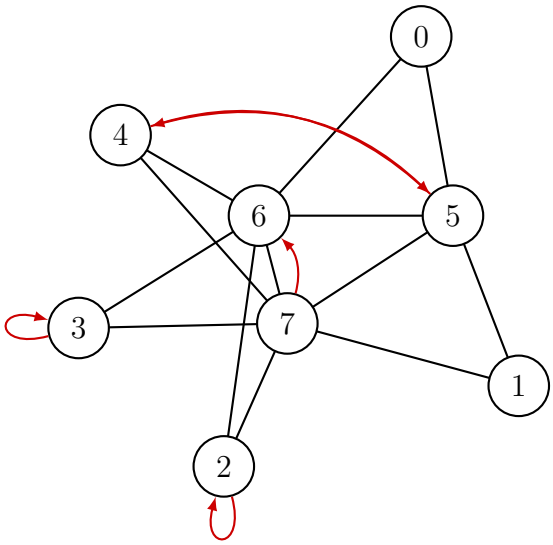
Sample 639 label: 0 (no extension)

$f(0)=4,\ f(1)=0,\ f(2)=3,$
$f(4)=1,\ f(5)=6,\ f(7)=7$



Sample 4060 label: 1 (extensible)

$f(0)=0,\ f(1)=4,$
$f(2)=5,\ f(3)=6$



Sample 3526 label: 0 (no extension)

$f(2)=2,\ f(3)=3,\ f(4)=5,$
$f(5)=4,\ f(7)=6$

Figure 4.3: Three selected examples of confident GIN failures corrected by GraphGPS. Node positions are computed by the NetworkX spring layout algorithm (seed 42). Red dashed arrows show f; self-loops mark fixed points.

4.7.2 Average mapping distance hypothesis

A specific hypothesis suggested by the GraphGPS architecture is that its Laplacian positional encoding lets it leverage long-range structural information that pure message passing cannot capture in a few layers. We operationalize this by defining, for each

test graph and its embedded partial automorphism f,

$$\overline{d}(G, f) \;=\; \frac{1}{|\mathrm{dom}(f)|} \sum_{v \in \mathrm{dom}(f)} d_G(v, f(v)),$$

the mean shortest-path distance between every source vertex and its image under f.

We then compare $\overline{d}$ between two groups:

- **Group A:** GraphGPS predicts correctly with $|p_{\mathrm{GPS}} - 0.5| > 0.3$ and GIN predicts incorrectly. ($n_A = 1207$)
- **Group B:** GIN predicts correctly. ($n_B = 5367$)

Summary statistics are reported in Table 4.5. The directional difference is consistent with the hypothesis: Group A has both a larger mean and a larger median mapping distance than Group B.

Table 4.5: Average mapping distance $\overline{d}(G, f)$ in the two groups.

Group	n	Mean	Median	Std
A (GraphGPS conf. correct, GIN wrong)	1207	1.278	1.143	0.809
B (GIN correct)	5367	1.170	1.000	0.832

We use a one-sided Mann–Whitney U test to check whether Group A values tend to be larger than Group B values. Unlike a t-test, this test does not assume the data follows a normal distribution; it works by ranking all values from both groups together and checking whether one group's values cluster higher in the ranking (see [30] for an accessible introduction to these tests and effect size measures). The test rejects the null hypothesis in favor of $\overline{d}_A > \overline{d}_B$ at any conventional significance level: $U = 3\,545\,392$, $p < 10^{-4}$ (U is the number of Group A/Group B pairs in which the Group A value is larger; p is the probability of observing a U this large if the two groups were drawn from the same distribution).

The rank-biserial correlation $r = +0.095$ is the effect size that accompanies this test. Intuitively, it is the probability that a randomly chosen Group A value exceeds a randomly chosen Group B value, minus the reverse probability. A value of 0 means the groups are indistinguishable; a value of 1 means Group A is always larger. The value $r = +0.095$ indicates a small but consistent shift.

A complementary measure, Cohen's d, expresses the same gap in units of standard deviations: $d = (\mathrm{mean}_A - \mathrm{mean}_B)/\text{pooled SD}$. Values below 0.2 are conventionally considered small. Here the mean difference of 0.108 is small relative to the within-group standard deviation of approximately 0.82, giving Cohen's $d \approx 0.13$. Mapping distance is therefore a statistically robust but practically modest signal: a real contributor to

GraphGPS's advantage, but not a complete account of it. The two distributions are shown in Figure 4.4.

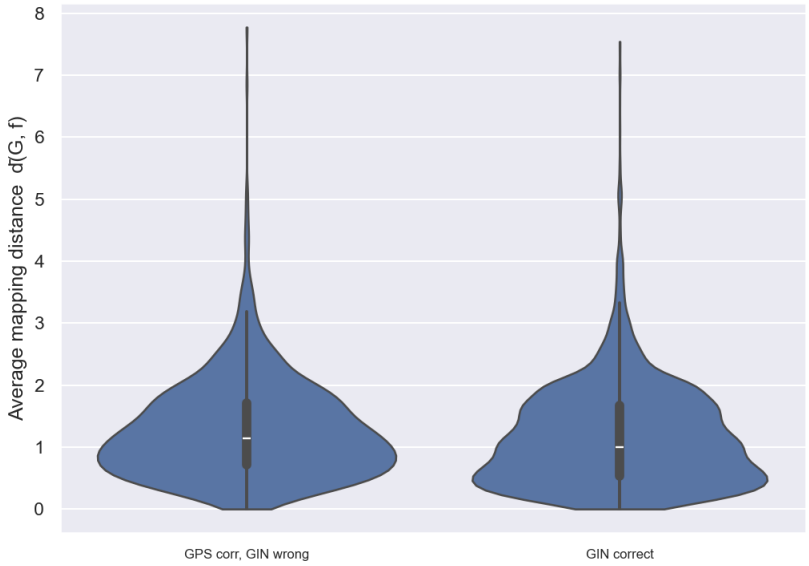


Figure 4.4: Violin plot of average mapping distance $\overline{d}(G, f)$ in Group A (GraphGPS confidently correct, GIN wrong) and Group B (GIN correct).

4.7.3 Role of graph diameter

A natural objection is that $\overline{d}$ may be large simply because the graph itself is "wide", in which case GraphGPS would be benefiting from a graph-level scale rather than from the specific positions of mapped vertices. To disentangle these we additionally compute the graph diameter $\mathrm{diam}(G)$ and the *relative* mapping distance $\overline{d}(G, f)/\mathrm{diam}(G)$.

Table 4.6: Diameter and relative mapping distance in the two groups.

Metric	Group	Mean	Median	U	p (one-sided)
Diameter	A	3.329	3.000	3 200 392	0.7494
	B	3.360	3.000		
Relative mapping dist.	A	0.421	0.417	3 564 470	< 0.0001
	B	0.382	0.364		

The two tests in Table 4.6 together give a clean answer. **Diameter alone does not differ between the groups** ($p = 0.75$, rank-biserial $r = -0.012$): the graphs that GraphGPS corrects are not systematically wider than the graphs GIN already gets right. **Relative mapping distance, on the other hand, is significantly larger in Group A** ($p < 10^{-4}$, $r = +0.100$, Figure 4.5). The signal is therefore not

about graph scale, it is about the position of the partial automorphism within the graph: GraphGPS corrects exactly those instances on which f moves vertices a longer distance relative to the graph's own scale.

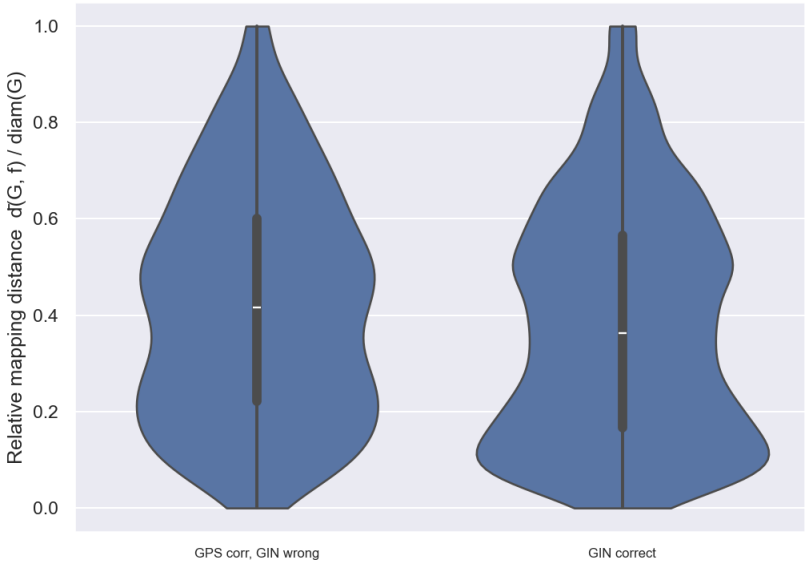


Figure 4.5: Violin plot of relative mapping distance $\overline{d}/\text{diam}$ in Group A and Group B. Group A shows a significantly larger relative distance ($p < 10^{-4}$, $r = +0.100$).

4.7.4 Stratification by diameter

To rule out the possibility that the relative-distance result is itself an artifact of mixing diameter buckets, we repeat the comparison within each diameter bucket. Results are in Table 4.7.

Table 4.7: Relative mapping distance, stratified by graph diameter. Δ *mean* is mean(A) − mean(B); the one-sided test is $H_1 : \overline{d}_A/\text{diam} > \overline{d}_B/\text{diam}$. Shaded rows have $n_A \leq 35$ and are dominated by sampling noise.

diam.	n_A	mean$_A$	n_B	mean$_B$	Δ mean	p (one-sided)
2	440	0.509	1875	0.459	+0.050	0.0001
3	288	0.484	1354	0.450	+0.034	0.0196
4	276	0.337	1217	0.297	+0.040	0.0005
5	134	0.238	584	0.217	+0.021	0.0297
6	35	0.312	178	0.279	+0.033	0.2570
7	10	0.209	43	0.309	−0.100	0.9064
8	10	0.345	44	0.216	+0.129	0.0243
10	8	0.176	40	0.182	−0.006	0.4450

For diameters 2 through 5, where Group A contains hundreds of samples per bucket, the relative-distance gap is positive and statistically significant in every bucket. The diameter-6, 7, 8, and 10 rows have only 8 to 35 samples in Group A and any single result there is dominated by sampling noise: the -0.100 at diameter 7 ($n_A = 10$) and the $+0.129$ at diameter 8 ($n_A = 10$) are not in tension; they are noise on either side of zero. The trustworthy conclusion is the one drawn from the dense buckets: even within graphs of identical diameter, GraphGPS-corrected instances place the partial automorphism on vertex pairs that are further apart in graph distance.

4.8 Discussion

Three findings emerge from the analysis.

1. **Marginal structural features explain little.** Within each model in isolation, none of `num_nodes`, `paut_size`, `paut_relative_size`, or `aut_grp_order` predicts errors with meaningful effect size. The Spearman correlations are all $|r| < 0.07$. This is consistent with the theoretical expectation that hardness for a 1-WL-bounded message-passing model is not reducible to a single summary statistic of the input.

2. **GraphGPS's residual errors concentrate on regular graphs.** Although GraphGPS reduces the error rate from 19.5% to 2.2%, the remaining errors are disproportionately on regular graphs (46% versus a population rate of 18%). Positional encoding does not fully resolve the symmetry-driven failure mode; it only mitigates it. A direct test rules out the sharpest version of the theoretical explanation: within regular graphs, vertex-transitive and non-vertex-transitive instances have statistically indistinguishable GPS error rates (4.12% vs 6.14%, $\chi^2 = 0.74$, $p = 0.39$), and the error concentration is carried by the larger non-vertex-transitive regular subgroup (41.6% of errors, 15.1% of the population). Regularity itself, rather than the specific LapPE injectivity failure on vertex-transitive graphs, is the operative correlate; its deeper cause remains open.

3. **GraphGPS specifically helps when mapped vertices are far apart relative to the graph's scale.** A direct paired comparison of the graphs on which the two models disagree shows a small but highly significant shift in $\overline{d}/\mathrm{diam}$. The shift is not driven by diameter alone (the groups match in raw diameter, $p = 0.75$) and persists when stratifying by diameter. This provides empirical support for the theoretical motivation of Laplacian PE: it is precisely the long-range, non-local structural information that pure message passing cannot transmit in a few layers.

Conclusion

This thesis used the partial automorphism extension problem as a controlled probe for the expressive power of graph neural networks. Building on the algebraic graph theory, the Weisfeiler-Leman color-refinement algorithm, and the message-passing framework reviewed in Chapter 1, we constructed a labeled benchmark from 4,985 House of Graphs base graphs [9]: roughly 82,000 training and 6,670 test instances, with positives obtained by orbit restriction and negatives generated by a Godsil–Kocay pseudo-similarity construction combined with a cross-orbit blocking step.

We then trained and analyzed two architectures: the Graph Isomorphism Network (GIN), which matches the 1-WL ceiling exactly, and GraphGPS, a hybrid model combining message passing, global self-attention, and Laplacian positional encodings.

GIN reaches a test accuracy of 0.8046 ($F_1 = 0.8195$); GraphGPS reaches 0.9777 ($F_1 = 0.9784$), a roughly 89% reduction in error rate (Tables 3.1, 3.2, and 4.2). The GIN ceiling proves robust both to explicit structural features and to a 1.5× increase in training data, isolating expressiveness as the bottleneck rather than feature quality or data volume.

The two models also differ structurally in how they fail. GIN's error rate stays at ≈ 0.21 across all non-trivial symmetry orders (Section 4.6.5), exposing the 1-WL barrier as a cliff: 1-WL-equivalent neighborhoods carry no information about the underlying automorphism group, regardless of its order. GraphGPS breaks through this cliff but the best single structural signal we find (corrected instances tend to have larger relative mapping distance $\overline{d}/\mathrm{diam}$, Section 4.7.2) has only a modest Cohen's $d \approx 0.13$, suggesting that the advantage accrues across many weak cues exposed jointly by global attention and positional encodings rather than from a single dominant feature.

Three caveats apply to the reported numbers. The test set is class-balanced for diagnostic purposes, whereas the natural distribution is dominated by non-extensibles. House of Graphs is a curated collection, so generalization to other distributions (random graphs, highly regular structures) is not guaranteed. Finally, the Laplacian positional encodings were precomputed once for the fixed corpus without sign-flip augmentation: eigenvector signs are consistent within the training and test sets but arbitrary across them. Applying the model to a new unseen graph runs the eigenvalue solver afresh, and the sign convention it chooses may differ from any training example. Because the

model was not trained with sign randomization, predictions on new graphs could be inconsistent without additional sign normalization or augmentation at inference time.

Summary of main contributions

- **Open benchmark dataset.** A labeled corpus of $\approx$ 82,000 training and 6,670 test instances of partial automorphism extension, built from 4,985 House of Graphs base graphs, with negatives generated by Godsil–Kocay pseudo-similarity and cross-orbit blocking so non-extensibility holds by construction (Proposition 1).
- **Empirical locating of the 1-WL bottleneck.** A GIN baseline reaches 80.46% test accuracy; controlled ablations show this ceiling is robust to four added structural features (Table 3.1) and to a 1.5$\times$ increase in training data (Table 3.2), which isolates expressiveness as the bottleneck.
- **Strong baseline beyond 1-WL.** GraphGPS with Laplacian positional encodings reaches 97.77% test accuracy, a roughly 89% reduction in error rate relative to GIN, showing that the task is within reach of hybrid message passing with global attention and spectral PE.
- **Structural failure-mode diagnosis.** Paired analyses identify the 1-WL cliff for GIN (Section 4.6.5), the over-representation of regular graphs in GraphGPS residual errors, and the relative mapping distance $\overline{d}/\mathrm{diam}$ (Section 4.7.2) as the structural signal most predictive of where the additional expressiveness pays off. The error concentration is on regular graphs, but a direct test shows that vertex-transitive and non-vertex-transitive regular graphs have statistically indistinguishable GPS error rates ($\chi^2 = 0.74$, $p = 0.39$); the concentration is carried by the larger non-vertex-transitive regular subgroup. Regularity itself is the operative correlate; its deeper cause remains open.

Several open directions follow. First, the cause of the regular-graph error concentration is not fully explained: a direct test rules out vertex-transitivity as the primary driver (error rates are similar inside and outside the vertex-transitive subclass, $p = 0.39$), so the relevant property of regular graphs remains open. Diagnosing it is a natural next step, and random node initialization (RNI, appending independently sampled Gaussian noise to each node's features before each forward pass) is a low-cost intervention worth testing, since it breaks any orbit symmetry stochastically regardless of the structural cause.

Second, the mapping-distance result points toward architectures that explicitly model vertex pairs rather than individual vertices, such as k-WL or subgraph GNNs [29], which can compare the neighborhoods of v and $f(v)$ directly.

Third, partial automorphism extension is a natural benchmark for follow-up expressiveness research: hardness is exactly characterized by WL-indistinguishability, the graph distribution is controlled, and the label is binary, which makes it easier to interpret than regression benchmarks on molecular property datasets.

Fourth, the MapleSAT line of work shows that a learned branching heuristic can accelerate a CDCL solver without sacrificing correctness [26]; an analogous question is whether a trained GNN could guide the orbit-partition refinement inside `nauty` or a similar backtracking automorphism search, substituting a learned policy for hand-crafted invariant orderings at decision points.

Bibliography

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

[2] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

[3] H. S. Baird and Y. E. Cho. An artwork design verification system. In *Proceedings of the 12th Design Automation Conference*, DAC '75, pages 414–420. IEEE Press, 1975.

[4] N. Biggs. *Algebraic Graph Theory*. Cambridge Mathematical Library. Cambridge University Press, 1993.

[5] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[6] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

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

[8] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[9] Kris Coolsaet, Sam D'hondt, and Jan Goedgebeur. House of graphs 2.0: A database of interesting graphs and more. *Discrete Applied Mathematics*, 325:97–107, 2023.

[10] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.

[11] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2020.

[12] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations, 2022.

[13] Matthias Fey, Jinu Sunil, Akihiro Nitta, Rishi Puri, Manan Shah, Blaž Stojanovič, Ramona Bendias, Alexandria Barghi, Vid Kocijan, Zecheng Zhang, Xinwei He, Jan Eric Lenssen, and Jure Leskovec. Pyg 2.0: Scalable learning on real world graphs, 2025.

[14] Mohammad Ghebleh, Salem Al-Yakoob, Ali Kanso, and Dragan Stevanović. Reinforcement learning for graph theory, I. Reimplementation of Wagner's approach, 2024.

[15] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[16] C. D. Godsil and W. L. Kocay. Constructing graphs with pairs of pseudo-similar vertices. *Journal of Combinatorial Theory, Series B*, 32(4):392–399, 1982.

[17] Martin Grohe. The Logic of Graph Neural Networks, January 2022. arXiv:2104.14624 [cs].

[18] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[19] Ningyuan Teresa Huang and Soledad Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8533–8537, 2021.

[20] C.A.M. Irniger. *Graph Matching: Filtering Databases of Graphs Using Machine Learning Techniques*. Dissertationen zur künstlichen Intelligenz. AKA, 2005.

[21] Robert Jajcay, Tatiana Jajcayova, Nóra Szakács, and Mária B. Szendrei. Inverse monoids of partial graph automorphisms, 2020.

[22] Charilaos I. Kanatsoulis, Evelyn Choi, Stephanie Jegelka, Jure Leskovec, and Alejandro Ribeiro. Learning efficient positional encodings with graph neural networks, 2025.

[23] R. Kimble, A. Schwenk, and P. Stockmeyer. Pseudosimilar vertices in a graph. *Journal of Graph Theory*, 5:171–181, 1981.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[25] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[26] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Theory and Applications of Satisfiability Testing – SAT 2016*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2016.

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[28] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of symbolic computation*, 60:94–112, 2014.

[29] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4602–4609, July 2019.

[30] Danielle J. Navarro. *Learning Statistics with R: A Tutorial for Psychology Students and Other Beginners*. Self-published, 2019. Available free online at `https://learningstatisticswithr.com`.

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

[32] Raffaele Paolino, Sohir Maskey, Pascal Welke, and Gitta Kutyniok. Weisfeiler and Leman Go Loopy: A New Hierarchy for Graph Representational Learning, November 2024. arXiv:2403.13749 [cs].

[33] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer, 2023.

[34] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations (ICLR)*, 2019.

[35] Ahsan Shehzad, Feng Xia, Shagufta Abid, Ciyuan Peng, Shuo Yu, Dongyu Zhang, and Karin Verspoor. Graph transformers: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–20, 2026.

[36] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[39] Adam Zsolt Wagner. Constructions in combinatorics via neural networks, 2021.

[40] Boris Weisfeiler and A. A. Lehman. A Reduction of a Graph to a Canonical Form and an Algebra Arising During This Reduction. *Nauchno-Technicheskaya Informatsia*, Ser. 2(N9):12–16, 1968.

[41] D.B. West. *Introduction to Graph Theory*. Featured Titles for Graph Theory. Prentice Hall, 2001.

[42] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks?, February 2019. arXiv:1810.00826 [cs].

[43] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[44] Chengxiao Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation?, 2021.

[45] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks, 2021.

Appendix: Source Code and Results

In the appendix of the electronic version of the thesis, you can find the source code of the program and files with the results of experiments. The source code is also published on the page `https://github.com/rivalxy/gnn-graph-theory`.