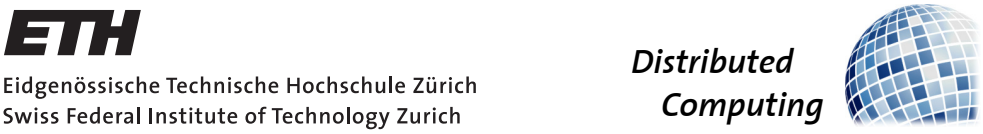


Adversarial Robustness of Expressive Graph Neural Networks

Bachelor's Thesis

Reto Merz

retmerz@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. Karolis Martinkus
Prof. Dr. Aleksandar Bojchevski
Prof. Dr. Roger Wattenhofer

August 22, 2023

Acknowledgements

Most of all I thank my advisor Karolis Martinkus for his expertise, guidance, and support throughout the thesis, and also for the freedom he left me in how to approach the project. Secondly, I would like to thank Aleksandar Bojchevski for his contributions to the project, which have shaped how I think about robustness. Additionally, I thank Professor Roger Wattenhofer and the Distributed Computing Group for making this thesis possible by providing the infrastructure to run the experiments. Finally, I would like to thank my family and friends who supported me all through my undergraduate studies.

Abstract

In recent years, graph neural networks (GNNs) have gained popularity as a tool for machine learning on graphs. Despite their success in many fields, many GNN architectures lack the expressiveness to differentiate between non-isomorphic graphs of certain classes, motivating more expressive architectures. Other works have shown basic GNN architectures and defenses vulnerable to adversarial attacks, which generate adversarial examples that are mispredicted by the model. We investigate how the expressivity of a GNN architecture influences its robustness to adversarial attacks by designing graph-level test-time attacks targeting the discrete structure of graphs with node and edge features. Our experiments indicate that PPGN, the most expressive architecture tested, is more robust on a range of attacks than the other tested architectures, and can even outperform adversarially trained models of other expressive architectures under attacks.

Contents

Chapter 1

Introduction and Related Work

In recent years, graph neural networks (GNNs) have gained popularity as a tool to apply machine learning to graph data and were successfully applied to a wide range of domains, such as social networks [1, 2], chemistry [3], drug discovery [4] and computer vision [5], exploiting structural information present in these fields. Despite their popularity, the basic models lack the expressiveness to differentiate between certain non-isomorphic graphs, such as k-regular graphs. To remedy the situation, a hierarchy of expressiveness based on how well GNNs can distinguish non-isomorphic graphs was introduced [6], and many novel architectures were proposed which provably exceed the expressiveness of the basic models and perform better in practice [7, 8, 9, 10, 11, 12].

As with any model deployed in the real world, GNNs are potentially exposed to adversarial attacks and distribution shifts. In a social network, a bad actor may try to determine a set of members to follow which minimizes his chance of being discovered to be malicious, i.e. find changes in the adjacency of the network graph that fool the model. In the molecular domain, an attacker might swap out certain atoms in a molecule, and thereby change the node features that are relayed to a model, to reduce its predicted toxicity. In general, attacks on graph models may target the adjacency of a graph as well as its node and edge features.

Similar to convolutional architectures for computer vision, unprotected GNNs were shown to be vulnerable to attacks. This led to a range of proposed defenses, which did not hold up to adaptive attacks [13]. However, these defenses are based on non-expressive GNNs, which raises the main questions this thesis tries to answer:

Are expressive GNNs more or less robust on graph-level tasks than less-expressive GNNs?

On the one hand, expressive models can represent more complex functions, which might enable them to learn more robust decision boundaries. On the other hand, they may be more sensitive to graphs that are not well represented in the training data, due to their improved capability to differentiate between graphs. However, even the basic GNNs should be able to distinguish between original and perturbed graphs for most perturbations.

To answer the question quantitatively, we run gradient-based evasion attacks on a range of expressive architectures. The selected architectures, which are introduced in more detail in Chapter 3, include DropGNN [7] and Equivariant Subgraph Aggregation Networks (ESAN) [9], which process multiple subgraphs to improve robustness,

Provably Powerful Graph Network (PPGN) [8] that builds higher-order representations of the adjacency matrix via matrix multiplication, and SignNet [10] which augments node features with eigenvector information of the graph Laplacian.

In the experiments, we examine the robustness of models from a given architecture that were trained in a supervised setting. This approach is concerned with models that arise in practice, and thus it cannot give insight into the robustness of the most robust model of a given architecture. Because expressiveness is established by proving the *existence* of model parameters that achieve certain properties, it is interesting to see whether models of more expressive architectures can express more robust decision boundaries. To research this aspect, we additionally test models obtained through adversarial training [14, 15], which is a technique to learn robust decision boundaries. Adversarial training trains the model on the strongest adversarial example in each training step instead of the original data. Its roots are in computer vision, but it has also been applied to GNNs [16]. Concerning expressive GNNs, we ask the following secondary questions:

How does the expressiveness of a model relate to robustness improvements due to adversarial training? Does the expressiveness of a model improve its performance on clean data when trained adversarially?

We expect positive answers to both questions because expressive GNNs are in theory better equipped to express robust decision boundaries, however, it is hard to say how big the impact will be in practice and how it will differ between expressive architectures.

Expressive GNNs are mostly applied on graph-level tasks with smaller graphs. We selected graph datasets that are commonly used to evaluate the performance of (expressive) GNNs:

- MolHIV from the Open Graph Benchmark (OGB) providing a binary classification task on small molecules [17].
- ZINC12k featuring a regression task on small molecules [18, 19].
- IMDB-BINARY, IMDB-MULTI and MUTAG from the TUDataset collection [20, 21], which are smaller datasets. IMDB-BINARY and IMDB-MULTI have binary and 3-way classification tasks on ego-graphs based on the collaboration of actors, and the task of MUTAG is to predict the mutagenic properties of small molecules.

The molecules are represented as graphs with node and edge features, giving additional information about the atoms and bonds. The graphs in the IMDB datasets however have no node or edge features. The MUTAG dataset was selected for the experiment on adversarial training due to its small size, which results in a reasonable training speed even with the high computational overhead of adversarial training.

To our knowledge, there are no studies on the robustness of expressive GNNs. However, the literature suggests a wide range of attacks on GNNs. The goal of an attack is to find a small perturbation of a graph such that the perturbed class gets misclassified in the case of a classification task or that the predicted value is far

from the labeled value in the case of a regression task. Due to the combinatorial nature of graphs, it might be infeasible to try all perturbations. This is why there has been a focus on gradient-based attacks, which relax the discrete structures of adjacency and features and optimize over real-valued tensors. Since many GNN architectures are not differentiable with respect to the adjacency matrix, differentiable surrogate models are required. A recent paper [13] suggests best practices for designing model-specific adaptive attacks. Xu et al. introduced a gradient-based attack on the adjacency of graphs that provides adversarial examples given a budget of allowed adjacency changes [22]. Nettack, presented in [23], considers models for node classification, which it attacks by modifying both the adjacency and node features in an unnoticeable way that keeps the node degree distributions similar. This attack greedily optimizes the adjacency and node features in alternating rounds and selects the next change based on gradient information from a surrogate model. None of these attacks consider attacking the edge features since their GNN architectures don't use edge features.

In Chapter 4, we present and adapt the topology attack of Xu et al. to graph-prediction models with edge features. We design an extension of this attack that additionally targets node and edge features besides the adjacency. We also briefly introduce some simple attacks that serve as baselines for the model's robustness as well as the performance of other attacks. Chapter 5 is devoted to the robustness experiments and their results. There, we give a detailed overview of our experimental setup, followed by an analysis of the generated data. We conclude in Chapter 6 by answering the posed questions.

Contributions

The contributions of this thesis are summarized as follows:

- We provide an overview of expressive GNN architectures.
- We design and implement differentiable surrogate models for popular expressive GNN architectures.
- We design, implement and evaluate gradient-based evasion attacks targeting the adjacency, node, and edge features considering specific threat models.
- We test the robustness of expressive GNN architectures on graph-prediction tasks under random and adversarial test-time perturbations.
- We evaluate how effective adversarial training is at improving the robustness of expressive GNNs and how it impacts the performance with clean data.

Chapter 2

Machine Learning on Graphs

Molecules, social networks, the control flow of programs, or other network-like structures can all be modeled by graphs, which are sets of nodes that are connected by binary edges. Vast amounts of data in these domains enable machine learning approaches to solve various tasks, such as predicting chemical properties of molecules, identifying spam accounts in social networks or even detecting bugs in code. This chapter introduces the basis of supervised machine learning approaches on graphs. As is typical for supervised machine learning they rely on a parametrized model architecture, which is fitted to a labeled dataset using a gradient-based optimization scheme like gradient descent and derivatives.

The fundamental property that models on graphs need to satisfy is *invariance under node permutations*, i.e. re-labeling the nodes of a graph should not change the answer of the model. The way many graph neural networks achieve this property is by restricting the first stage of processing to node-local operations and then, in the case of graph-level tasks, continuing computation on an aggregate over all nodes.

2.1 Graph Data

Definition 2.1 (undirected graph)**.** An *undirected graph* is tuple (V, E) for some finite set of nodes $V = \{v_0, \ldots, v_{n-1}\}$ and edges $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$. For convenience, we use $N := |V|$ and $M := |E|$.

We will often represent the edge relation as a sparse *edge index tensor* $I \in \{0, \ldots, N-1\}^{2\times 2M}$, where $\{v_i, v_j\} \in E \iff I_{:,k} = [i, j] \wedge I_{:,l} = [j, i]$ for some $k, l \in \{0, \ldots, 2M-1\}$. The edge index tensor represents each undirected edge as two directed edges and fixes an ordering of the directed edges. A dense way[1] to represent the edges is via the *adjacency matrix* $A \in \{0, 1\}^{N\times N}$, defined by $A_{i,j} = 1 \iff \{v_i, v_j\} \in E$.

Depending on the machine learning task, we might have additional information about the kinds of nodes and edges present in the graph, e.g. atom and bond information of a molecule, resulting in an *attributed graph* $G = (V, E, X, Y)$. The *node features* $X : V \to \mathbb{R}^{d_n}$ map each node to a vector in $\mathbb{R}^{d_n}$, which in a sparse encoding under a fixed node order is represented as a tensor in $\mathbb{R}^{N\times d_n}$. Similarly, the *edge*

[1] Meaning that the adjacency information between each pair of nodes is explicitly represented.

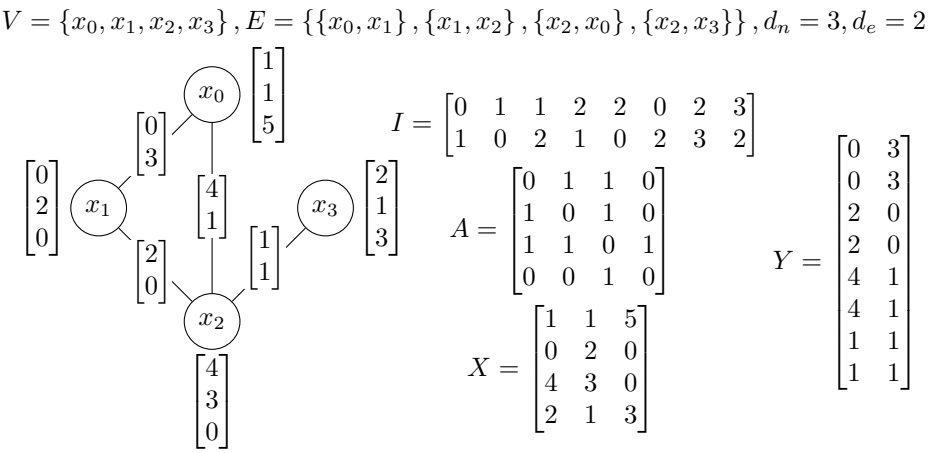


Figure 2.1: Example of a graph with node and edge features, including edge index tensor, node and edge feature tensors, and adjacency matrix.

features $Y : E \to \mathbb{R}^{d_e}$ map each edge to a vector in $\mathbb{R}^{d_e}$, which is sparsely represented as a tensor in $\mathbb{R}^{2M \times d_e}$. The dense representation of edge features is given by a tensor in $\overline{Y} = \mathbb{R}^{N \times N \times d_e}$, where $\overline{Y}_{i,j} = Y(\{i,j\})$ for all $\{i,j\} \in E$ and $\overline{Y}_{i,j} = \mathbf{0}$ if $\{i,j\} \notin E$. We use $\mathcal{G}$ to denote the set of all attributed graphs.

Usually, the sparse representation of graphs is preferred since it allows for faster processing of sparse graphs (where $M \ll \binom{N}{2}$). The dense representation of an attributed graph, given by the adjacency matrix and the dense edge feature tensor, will be important for the attacks because it describes the adjacency between any pair of nodes as a real number, for both positive ($\in E$) and negative ($\notin E$) edges. This real-valued adjacency structure is the target of the gradient-based attacks.

Feature Embeddings

The node and edge features often only take a finite number of values and encode a class instead of a value. Discrete feature vectors in $\mathbb{N}^d$ get embedded into a real feature vector in $\mathbb{R}^{d'}$. The embedding of $x \in \mathbb{N}^d$ is computed as $\sum_{i=1}^{d} L_i(x_i)$, where $L_i : \{0, \ldots, k_i - 1\} \to \mathbb{R}^{d'}$ is a learned lookup table (called an *embedding* in PyTorch) for the i-th feature.

Embeddings can be computed globally (once) for a model or locally per submodule. Although embeddings computed per submodule are more powerful[2], we only use global embeddings in this thesis due to the convenience of implementation for gradient attacks and the small difference in practice to local embeddings.

[2]Any model with global embeddings can be represented by a model with local embeddings where each local embedding computes the global embedding.

Mini-Batching

During model training, the gradient on the loss of the entire dataset is estimated by the gradient of the loss of just a small subset of the data, a so-called *mini-batch*. Mini-batching may also result in higher GPU utilization, which speeds up processing. For the sake of completeness and understanding of implementation details, we briefly illustrate what shape the batched representations of sparse and dense graphs have.

Sparse graphs are batched by combining them into a large graph by relabeling the nodes to avoid conflicts and then joining all updated nodes and edges. The components of the batched graph are the original graphs. Given a batch of B sparse graphs with node counts $N_1, \ldots, N_B$ and edge counts $M_1, \ldots, M_B$, the batched versions of graph-representing tensors have the following shapes:

node features	$\mathbb{R}^{(N_1+\ldots+N_B)\times d_n}$
edge index	$\mathbb{R}^{2\times 2(M_1+\ldots+M_B)}$
edge features	$\mathbb{R}^{2(M_1+\ldots+M_B)\times d_e}$

Dense graphs are batched by joining the graph tensors along a new dimension. In case the graphs have a differing number of nodes, the tensors are zero-padded to the largest number of nodes per graph in the batch (along node-labeled dimensions). A batch of B graphs is represented by dense tensors of shapes:

node features	$\mathbb{R}^{B\times N\times d_n}$
adjacency matrix	$\{0,1\}^{B\times N\times N}$
edge features	$\mathbb{R}^{B\times N\times N\times d_e}$

Graph-ML tasks

The labeling of the data depends on the type of task we want to solve. Machine learning tasks on graphs can be categorized as node prediction, link prediction, and graph prediction. An example of a node prediction task is the detection of bad agents in social networks, where we wish to assign each participant (represented by a node in the social network graph) a credibility score. However, this thesis focuses on graph prediction tasks, which means that our graph data contains a per-graph label. Typical graph prediction tasks in the chemical domain are to predict the chemical properties of molecules.

2.2 Message Passing Neural Networks

Message Passing Neural Networks (MPNNs) start with initial node representations provided by the data. These node representations are repeatedly updated based on the representation of neighboring nodes and the corresponding edges in a series of layers. A single layer consists of three steps: each node passes a message to its neighbors (1), which then aggregate all received messages (2) and finally update their representation based on this aggregate (3). In the case of graph-level tasks, the node representations obtained from multiple layers of message-passing layers are pooled together (often using sum, mean, or max aggregation) to obtain a representation of

the entire graph. This graph-level representation is typically further processed by a multi-layer perceptron (MLP).

In the most general form, a MPNN layer is described by the following formula[3]:

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{y}_{j,i} \right) \right), \tag{2.1}$$

where $\mathbf{x}_i^{(k)} \in \mathbb{R}^d$ denotes the feature vector of node i in the k-th layer of the network and $\mathbf{y}_{i,j}$ the feature vector of the edge connecting node i and j[4]. The messages $\phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right)$ (step 1) of all neighbors $j \in \mathcal{N}(i)$ of node i get aggregated by a permutation invariant operator $\bigoplus_{j \in \mathcal{N}(i)}$ (step 2) such as $\sum$ (sum), $\frac{1}{|\mathcal{N}(i)|} \sum$ (mean) or max. Finally, $\gamma^{(k)}$ computes the new representation of node i based on its current representation and neighborhood aggregate (step 3). $\phi^{(k)}$ and $\gamma^{(k)}$ are parametrized functions that may be learned during training.

This layer is compatible with batched graphs (where multiple graphs are represented as one big graph) since every node has the same neighborhoods in the batched graph as in the original graph. The global graph pooling step however has to be adjusted to the batched graph as only nodes from the same original graph should be aggregated. PyTorch Geometric provides functions that implement this behavior.

The final graph level MPNN $F : \mathcal{G} \to \mathbb{R}^n$ has the form

$$F = r \circ p \circ l_k \circ \ldots \circ l_1, \tag{2.2}$$

consisting of a series of k MPNN layers $l_1, \ldots, l_k : \mathcal{G} \to \mathcal{G}$ (equation (2.1)), a global pooling operation $p : \mathcal{G} \to \mathbb{R}^m$ and a readout MLP $r : \mathbb{R}^m \to \mathbb{R}^n$.

As mentioned earlier, an important property that (graph-level) GNNs need to satisfy is invariance under node relabeling. Consider an attributed graph $G = (V, E, X, Y)$ on N nodes with node features $X : V \to \mathbb{R}^{d_v}$ and edge features $Y : E \to \mathbb{R}^{d_e}$. The group of node permutations S_N acts on G as follows

$$\begin{aligned} G' = (V, E', X', Y') &= \sigma \cdot G \\ \text{where } \{v_i, v_j\} \in E' &\iff \left\{ v_{\sigma^{-1}(i)}, v_{\sigma^{-1}(j)} \right\} \in E, \\ X'(v_i) &= X(v_{\sigma^{-1}(j)}), \\ Y'\left(\{v_i, v_j\}\right) &= Y\left(\left\{ v_{\sigma^{-1}(i)}, v_{\sigma^{-1}(j)} \right\} \right). \end{aligned}$$

MPNNs layers $l_i : \mathcal{G} \to \mathcal{G}$ are node-permutation equivariant in the sense that $l_i(\sigma \cdot G) = \sigma \cdot l_i(G)$ for any $G \in \mathcal{G}$ and $\sigma \in S_N$, since $v_{\sigma(j)} \in \mathcal{N}_{\sigma \cdot G}(v_{\sigma(i)}) \iff v_j \in \mathcal{N}_G(v_i)$. The global graph pooling $p : \mathcal{G} \to \mathbb{R}^m$ is node-permutation invariant as it satisfies $\forall G \in \mathcal{G}\ \forall \sigma \in S_N . p(G) = p(\sigma \cdot G)$. Since the MPNN layers propagate node permutations (equivariance) and the pooling cancels out node permutations (invariance), any MPNN $F = r \circ p \circ l_k \circ \ldots \circ l_1$ is node-permutation invariant.

[3] This description is taken from the PyTorch Geometric documentation [24].

[4] Note that the edge representations are not updated.

Graph Isomorphism Network

The Graph Isomorphism Network (GIN) was introduced in [6] as a simple architecture that is maximally expressive for graph classification among the standard GNNs (more on this in the next chapter). It is a special case of a general MPNN in which each layer computes

$$\mathbf{x}_i^{(k)} = h^{(k)}\left((1+\varepsilon)\mathbf{x}_i^{(k-1)} + \sum_{j\in\mathcal{N}(i)} \mathbf{x}_j^{(k-1)}\right), \tag{GINConv}$$

where $h^{(k)}$ is a trainable MLP and ε is typically fixed to 0. Due to its similarity to convolution (where the next representation of some element is a weighted sum over its neighbors), this layer is called *GINConv*.

It can be easily adapted to also use edge features, resulting in the *GINEConv* operator, which was introduced in [25]:

$$\mathbf{x}_i^{(k)} = h^{(k)}\left((1+\varepsilon)\mathbf{x}_i^{(k-1)} + \sum_{j\in\mathcal{N}(i)} \mathrm{ReLU}(\mathbf{x}_j^{(k-1)} + \mathbf{e}_{j,i})\right). \tag{GINEConv}$$

It is important that the node feature dimension d_n matches the edge feature dimension d_e in every layer as $\mathbf{x}_j^{(k-1)} + \mathbf{e}_{j,i}$ is not defined otherwise.

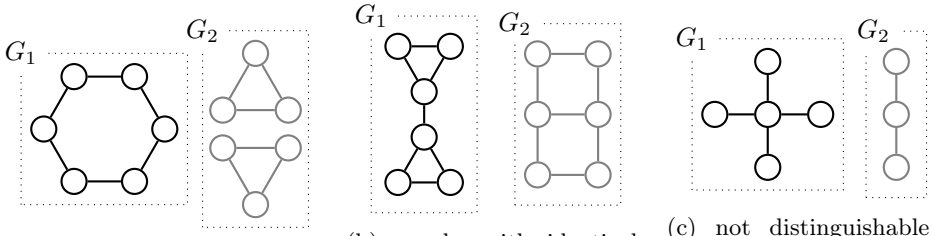
Chapter 3

Expressive Graph Neural Networks

As it turns out, the node-local way of handling graphs by MPNNs is not very *expressive*, as the basic models cannot even decide graph connectivity. The literature suggests many architectures which improve upon the poor expressiveness and are provably more expressive in a hierarchy of Weisfeiler-Lehman isomorphism tests. This chapter defines how the expressiveness of models is quantified in the literature and presents various architectures that are provably more expressive than MPNNs.

3.1 Expressiveness and the Weisfeiler-Lehman isomorphism test

A classic example (Fig. 3.1 a) of non-isomorphic graphs that cannot be told apart by MPNNs is a $2k$-cycle and two k-cycles (where all nodes and edges have the same features). This happens because all nodes see two neighbors with the same labels in every neighborhood aggregation step, which results in the representation for each node after each layer. This example illustrates that the expressive power of graph neural networks is limited by the neighborhood that is observed in every aggregation step. Indeed, a single 6-cycle and two 3-cycles can be told apart by comparing 2-



(a) k-regular graphs are not distinguishable by MPNNs.

(b) graphs with identical degree distributions (from [7], Fig. 2b)

(c) not distinguishable by mean neighborhood aggregation models

Figure 3.1: Pairs of graphs (G_1, G_2) that are not distinguishable by various GNN architectures.

hop neighborhoods, since the 2-hop neighborhoods of nodes in a 6-cycle are paths of length 3, whereas the 2-hop neighborhoods for the 3-cycles are triangles. Another example (Fig. 3.1 b, taken from [7]) again illustrates the limited expressive power of MPNNs, which fail to distinguish the two graphs since they have the same node-degree distribution (two nodes of degree 3 and four nodes of degree 2).

WL test The two presented examples hint towards the usefulness of degree distributions to classify the expressive power of graph neural networks. They are an important ingredient in the Weisfeiler-Lehman isomorphism test (WL test) [26], which is an efficient algorithm that approximately solves the hard graph-isomorphism problem for which we know no polynomial time algorithm [27]. The idea is to initially color each node by its features such that nodes with the sane feature get the same color,[1] and then iteratively update each node's color based on the color distribution among its neighbors until a fixed point is reached. In case the final color distributions between the two graphs differ, we can conclude that the graphs are not isomorphic. This algorithm is also known as 1-WL test.

k-WL test The extended family of k-WL tests follows a similar scheme as the 1-WL test but colors k-tuples of nodes instead of coloring single nodes. This improves the precision of the algorithm in the sense that for any k it is possible to construct two non-isomorphic graphs that can be differentiated by $k+1$-WL but not by k-WL for $k \geq 2$. 1-WL has the same power as 2-WL. [28, 8]

A precise formulation of the k-WL test taken from [8] is as follows: We denote the color of node-tuple $n \in V^k$ in round l by C^l_n. Initially, the color C^0_n of n is given by its isomorphism type, meaning that two tuples $n, n' \in V^k$ get the same color if $n_i = n_j \iff n'_i = n'_j$, $\deg(n_i) = \deg(n'_i)$ and $\{n_i, n_j\} \in E \iff \{n'_i, n'_j\}$ for all $i, j \in \{1, \dots, k\}$.

The j-th neighborhood $N_j(n)$ of node n, along which colors are updated, consists of all k-tuples where the j-th entry is given by an arbitrary node and the others are as in n, i.e.

$$N_j(n) = \left\{ n' \mid n'_j \in V, n'_1 = n_1, \dots, n'_{j-1} = n_{j-1}, n'_{j+1} = n_{j+1}, \dots, n'_k = n_k \right\}.$$

The color update for each round is

$$C^l_n = \text{hash}\left(C^{l-1}_n, \left(\left\{ \left\{ C'^{l-1}_n \mid n' \in N_j(n) \right\} \right\} \mid j \in \{1, \dots, k\} \right) \right),$$

where $\{\{\cdot\}\}$ denotes a multiset and hash is an injective function.

This notion of expressiveness can be applied to GNNs in the following way:

Definition 3.1 (k-WL-expressiveness)**.** A GNN architecture is called **k-WL-expressive** if for all $G, G' \in \mathcal{G}$ that can be distinguished by the k-WL graph isomorphism test, there exists a model M in the architecture such that $M(G) \neq M(G')$.

Xu et al. [6] showed that MPNNs are at most as expressive as the 1-WL test. They designed the GIN(E) architecture, introduced in the previous chapter, to specifically be 1-WL-expressive.

[1] In case the nodes have no features (as in the above examples), each node is initially assigned the same color.

Architecture	WL-expressiveness	Complexity
meanGIN(E)	less than 1-WL	$\mathcal{O}(N)$
GIN(E)	1-WL	$\mathcal{O}(N)$
BasisNet	more than 1-WL	$\mathcal{O}(N)$
DropGIN(E)	more than 1-WL	$\mathcal{O}(rN), r \approx N$
DSS-GNN with EGO+	more than 1-WL	$\mathcal{O}(N^2)$
PPGN	3-WL	$\mathcal{O}(N^3)$

Table 3.1: WL-hierarchy and complexity of expressive GNN architectures

SUM vs MEAN neighborhood aggregation

As noted in [7], the neighborhood aggregation type can influence the WL-expressiveness of a model. Models with mean or max neighborhood aggregation generally don't achieve 1-WL-expressiveness, whereas models with sum aggregation have better expressiveness. Figure 3.1 c shows two graphs where initially equal node features remain equal over a series of MPNN layers. We call GIN(E) models with mean neighborhood aggregation meanGIN(E). Each meanGINE layer computes

$$\mathbf{x}_i^{(k)} = h^{(k)}\left((1+\varepsilon)\mathbf{x}_i^{(k-1)} + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} \mathrm{ReLU}(\mathbf{x}_j^{(k-1)} + \mathbf{e}_{j,i})\right).$$

Although meanGINE models are not 1-WL-expressive, we intuitively expect them to be more robust against adjacency changes because the average over a set of values is less impacted when an element is added or removed than the sum.

Expressiveness in Practice

Although the notion of WL-expressiveness is powerful to compare GNNs, it doesn't provide enough resolution to compare any two architectures as there exist models that are more than k-WL-expressive[2] but not $k+1$-WL-expressive. Interestingly, there exist natural extensions of GNNs that achieve k-WL-expressiveness for any $k \in \mathbb{N}$ [11, 8]. However, these highly-expressive architectures scale badly both in terms of running time and memory, leading to a trade-off between expressivity and computational effort. k-WL expressiveness is generally associated with an $\mathcal{O}(N^k)$ running time due to the N^k node-tuples of size k. Considering these points, table 3.1 summarizes the expressiveness and time complexity of the GNN architectures considered in this thesis.

3.2 Equivariant Sub-graph Aggregation Network

Equivariant Sub-graph Aggregation Networks (ESAN) are GNN architectures introduced by [9], whose main feature is the processing of bags of subgraphs, which enables higher expressiveness.

[2]Meaning that they can differentiate graphs that are not differentiable by k-WL.

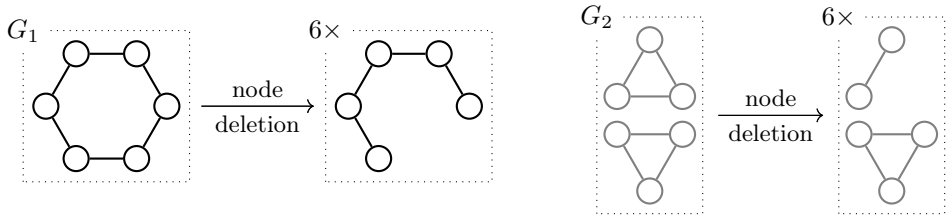


Figure 3.2: To motivate the expressive power of ESAN, consider the following example: Under a single node deletion, the multiset of subgraphs of a 6-cycle (G_1) and two triangles (G_2) can be distinguished by the 1-WL test. The multiset of subgraphs of G_1 are paths with five nodes, where the 1-WL algorithm will converge to assign different colors to the middle node, the two outer nodes, and the remaining nodes. In contrast, the converged 1-WL algorithm only assigns two distinct colors to nodes in the subgraphs of G_2, one for the triangle and one for the connected pair of nodes.

For the design of the architecture, the authors faced two main challenges: creating a network that is equivariant under the order of subgraphs (1) and determining suitable subgraph selection strategies that achieve high expressiveness and perform well in practice (2).

For (1), the authors decide to represent a bag (multiset) of m subgraphs of a graph with N nodes as an adjacency tensor $\mathbb{R}^{N\times N\times m}$ and node feature tensor $\mathbb{R}^{N\times d_v\times m}$.

They define the group $H = S_m \times S_N$, under which the feature encoder should be equivariant. An element $(\tau, \sigma) \in H$ is interpreted as permutation on the subgraphs τ and a permutation on the node order σ and thus acts on the graph tensors as follows:

$$((\tau,\sigma)\cdot A)_{i,j,k} = A_{\sigma^{-1}(i),\sigma^{-1}(j),\tau^{-1}(k)}$$
$$((\tau,\sigma)\cdot X)_{i,j,k} = X_{\sigma^{-1}(i),j,\tau^{-1}(k)}.$$

Then they introduce multiple architectures that are H-invariant, based on multiple H-equivariant layers. An H-equivariant layer L updates the node feature tensor of the i-th subgraph as follows

$$(L(A,X))_i = L^1\left(A_i, X_i\right) + L^2\left(\sum_{j=1}^m A_j, \sum_{j=1}^m X_j\right),$$

where (A_j, X_j) is the representation of the j-th subgraph and L^1, L^2 are arbitrary graph encoders such as MPNN layers. Instead of the sums in L^2, other permutation invariant aggregation operators are possible, such as the mean or the max.

The DSS-GNN architecture presented in the ESAN paper consists of three parts, similar to MPNN architectures:

$$F_{\text{DSS-GNN}} = E_{\text{sets}} \circ R_{\text{subgraphs}} \circ E_{\text{subgraphs}}.$$

- $E_{\text{subgraphs}} : \mathbb{R}^{N\times N\times m} \times \mathbb{R}^{N\times d_v\times m} \to \mathbb{R}^{N\times N\times m} \times \mathbb{R}^{N\times d'_v\times m}$ encodes each subgraph based on H-equivariant layers
- $R_{\text{subgraphs}} : \mathbb{R}^{N\times N\times m} \times \mathbb{R}^{N\times d'_v\times m} \to \mathbb{R}^{d'_v\times m}$ aggregates the node features per subgraph, and

- $E_{\text{sets}} : \mathbb{R}^{d'_v \times m} \to \mathbb{R}^{d''_v}$ is a permutation invariant set encoder, such as the mean over the subgraphs as in the implementation.

The paper also presents the DS-GNN architecture, which we don't consider because DSS-GNN is at least as expressive as DS-GNN and DS-GNN tends to perform worse in practice, based on the experiments of the authors.

For the subgraph selection (2), the authors consider four strategies. The **Node-Deleted (ND)** policy considers all subgraphs obtained by deleting a single node, which is implemented by removing all of its adjacent edges. Similarly, the **Edge-Deleted (ED)** policy creates all subgraphs obtained by deleting a single edge. The **EGO** policy generates k-hop (ego) subgraphs centered at each node. Finally, **EGO+** has the same subgraphs as EGO in terms of adjacency but augments the node features to mark the central node of each ego subgraph.

The authors prove that the ED subgraph selection policy is the most expressive in terms of distinguishing strongly regular graphs, although the ND and EGO+ (with N-hop subgraphs) policies are also able to differentiate between different classes of strongly regular graphs[3]. In practice, EGO+ seems to be the best-performing policy based on the experiments of the authors, with a significant difference in performance on ZINC12k and a within-variance difference on the MolHIV and the IMDB datasets.

The number of subgraphs, which is N for the ND and EGO+ policies and M for the ED policy, grows with the size of the graph. To improve the scalability of the ESAN architectures, the authors suggest subgraph sampling, where only a randomly chosen sub(multi)set of graphs is used. They evaluate this strategy with sampling fractions 50%, 20%, and 5% on the TUDatasets and MolHIV and find similar performance to using the full bag of subgraphs.

3.3 DropGNN

DropGNN [7] follows a similar strategy as ESAN to improve its expressiveness, namely by processing a set of subgraphs. Whereas ESAN constructs subgraphs deterministically, DropGNN does so in a randomized manner, by independently removing (dropping out) each node with probability p. The probability p should be chosen such that each subgraph (also called *run*) observes a different d-hop neighborhood (over d layers of message passing), but the runs should be similar enough so that the model output doesn't fluctuate too much due to different nodes being dropped out. Ideally, there is exactly one dropped-out node in every neighborhood of interest $\Gamma \subseteq V$ around each node. Thus the authors optimize p such that the probability of a 1-dropout among $\gamma := |\Gamma|$ is maximal, which is when

$$p = \frac{1}{1+\gamma}.$$

They also show that when using $r \geq \Omega(\gamma)$ runs, the expected number of observations of each 1-dropout in Γ is at least 1.

The runs are processed independently over multiple MPNN layers and the resulting node embeddings are averaged over all runs before global pooling and readout.

[3] This is not possible with 1-WL

3.4 Provably Powerful Graph Network

The PPGN architecture [8] operates on a dense representation $\boldsymbol{X} \in \mathbb{R}^{N \times N \times a}$ of the input graph. Given a graph G, this representation is derived as follows: The edges of G are represented by their adjacency matrix $A \in \mathbb{R}^{N \times N \times 1}$. In case G has node features $X : V \to \mathbb{R}^{d_v}$, they are represented by $X' \in \mathbb{R}^{N \times N \times d_v}$, where

$$X'_{i,j} = \begin{cases} X(v_i) & \text{if } i = j \\ \mathbf{0} \in \mathbb{R}^{d_v} & \text{else} \end{cases}.$$

Edge features $Y : E \to \mathbb{R}^{d_e}$ are represented by the dense edge features $Y' \in \mathbb{R}^{N \times N \times d_e}$, defined by

$$Y'_{i,j} = \begin{cases} Y(\{i,j\}) & \text{if } \{i,j\} \in E \\ \mathbf{0} \in \mathbb{R}^{d_e} & \text{else} \end{cases}.$$

The initial graph representation is the concatenation of these three[4] tensors along the last axis, i.e. $\boldsymbol{X} = (A, Y', X') \in \mathbb{R}^{N \times N \times (1+d_v+d_e)}$ for the input to the model.

The PPGN architecture updates this representation in a series of *blocks*. A single block applies three MLPs $m_1, m_2 : \mathbb{R}^a \to \mathbb{R}^b$ and $m_3 : \mathbb{R}^a \to \mathbb{R}^{b'}$ independently to each feature of $\boldsymbol{X}$. Denote the obtained tensors by $B, B' \in \mathbb{R}^{N \times N \times b}$ and $C \in \mathbb{R}^{N \times N \times b'}$. The block then performs matrix multiplication between each of the b features of B and B' to obtain a tensor $W \in \mathbb{R}^{N \times N \times b}$, given by $W_{:,:,k} = B_{:,:,k} \cdot B'_{:,:,k}$. The block output is then given by the concatenation $(C, W) \in \mathbb{R}^{N \times N \times (b+b')}$.

The full model architecture $F = m \circ h \circ B_d \circ \ldots \circ B_1$ consists of d such blocks $B_1, \ldots, B_d$, an invariant feature extraction layer $h : \mathbb{R}^{N \times N \times b} \to \mathbb{R}^{b'}$ and a readout MLP $m : \mathbb{R}^{b'} \to \mathbb{R}^{b''}$. An option for the invariant layer h is the sum over the first two dimensions, however, our implementation combines this sum with trace information, processed by individual MLPs.

To motivate the expressiveness of PPGN, the authors refer to the power of adjacency matrix exponentiation to compute interesting graph properties and take up the example of a 6-cycle and two disjoint 3-cycles (Fig. 3.1 a). They note that a PPGN model with three blocks is capable of computing A^3 (e.g. by setting m_1, m_2 and m_3 in each block to the identity) and to subsequently read out trace(A^3). It is well known that trace(A^3) counts the number of triangles in a graph. A 6-cycle has no triangles, whereas the graph with two disjoint 3-cycles has 2, which allowed PPGN to differentiate between the two graphs. Besides this nice intuition, the authors prove that PPGN is at least as expressive as 3-WL.

3.5 SignNet

SignNet, proposed in [10], achieves a higher expressiveness by augmenting the node features with eigenvector information of the graph Laplacian matrix. The eigenvectors of the graph Laplacian encode structural information about its graph, e.g. about connectivity, clusters, and how close nodes are [29].

[4] Or less in case G has no node or edge features.

Definition 3.2 (Normalized Graph Laplacian)**.** Given a graph on N nodes with adjacency matrix $A \in \{0,1\}^{N\times N}$ and node degree matrix $D = \mathrm{diag}(\deg(v_1), \dots, \deg(v_N))$, its *normalized graph Laplacian* is given by

$$L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}.$$

Note that L is symmetric if A is symmetric. Since A is symmetric for the graphs considered in this thesis, this means that there exist orthonormal eigenbases of L. Even after ordering the normalized eigenvectors by the size of the respective eigenvalue, there remain multiple ways to encode the eigenvalue information of L:

- Every eigenvector has a *sign symmetry*: If v is an eigenvector of A, then so is $-v$ (since $A(-v) = -Av = -\lambda v = \lambda(-v)$).
- *Basis symmetry*: If there is an eigenvalue with geometric multiplicity $k > 1$, the associated eigenvectors $v_1, \dots, v_k$ may be reordered in any way or replaced by any other orthonormal basis of span $\{v_1, \dots, v_k\}$. If the columns of $V = \begin{bmatrix} v_1 & \dots & v_k \end{bmatrix} \in \mathbb{R}^{N\times k}$ are an eigenbasis for some shared eigenvalue, then so are the columns of VQ for any orthogonal $Q \in \mathbb{R}^{k\times k}$.

In the paper, two architectures are presented. SignNet only achieves sign-invariance and BasisNet achieves both sign- and basis-invariance. Since the authors only implement SignNet, we will not further discuss BasisNet.[5]

The following theorem is fundamental for the SignNet architecture:

Theorem 3.3 (Sign-Invariant Functions)**.** *A function $f : \mathbb{R}^N \to \mathbb{R}^{d_{enc}}$ is sign-invariant if and only if there exists a function $\phi : \mathbb{R}^N \to \mathbb{R}^{d_{enc}}$ such that $f(x) = \phi(x) + \phi(-x)$ for all $x \in \mathbb{R}^N$.*

So we can encode any vector $v \in \mathbb{R}^N$ in a sign-invariant manner as $\phi(v) + \phi(-v)$ for any ϕ. To encode multiple eigenvectors $v_1, \dots, v_k \in \mathbb{R}^N$, the authors propose the sign invariant network $f : \mathbb{R}^{N\times k} \to \mathbb{R}^{d_{\text{out}}}$ as

$$f(v_1, \dots, v_k) = \rho\left([\varphi(v_i) + \varphi(-v_i)]_{i=1}^k\right).$$

for neural networks ρ, φ and vector concatenation $[\cdot]_i$. This form can compute any continuous function that is sign-invariant to each $v_1, \dots, v_k$ (under mild conditions).

To make f permutation equivariant under S_N, $\varphi : \mathbb{R}^N \to \mathbb{R}^N$ and $\rho : \mathbb{R}^{N\times k} \to \mathbb{R}^{N\times k'}$ have to be permutation equivariant. The authors suggest element-wise MLPs or MPNN layers.

The complete SignNet architecture first computes sign-invariant positional encodings for each node with f, using the first k eigenvectors with positive eigenvalues[6], ordered by their eigenvalue. The positional encodings are added to the existing node features. Then, the new graph is processed using a GNN.

[5] It is interesting to note that a large fraction of real-world graphs do contain higher-dimensional eigenspaces. In ZINC12k, 64.1% of graphs have a multiplicity of at least 1 and in MolHIV even 68.0% of graphs do, as pointed out in the paper.

[6] The eigenvector with eigenvalue 0 is discarded.

Chapter 4

Robustness and Attacks

The robustness of a machine learning architecture vaguely corresponds to how easy it is to steer its predictions in a certain direction, leading to misclassified inputs or deviations from regression labels. An evasion attack deliberately exploits the vulnerability of missing robustness and produces *adversarial examples*, which are inputs into a trained model that it predicts badly in case the attack was successful.

In the literature, both poisoning (training-time) attacks and evasion (evaluation-time) attacks are studied [30, 13]. Poisoning attacks assume that the adversary can sabotage the training data, which is especially interesting for models that are continuously trained in an online fashion. Due to memory constraints, we focus exclusively on evasion attacks in this thesis.[1]

In this chapter, we first define the attack model and then proceed to demonstrate some basic attacks. These attacks assume that the attacker can use an evaluation oracle to obtain the output logits of the model for a considerable number of inputs. Subsequently, we explore gradient-based attacks, which require a more potent attacker with access to the model parameters.

4.1 Adversarial Attacks

In case a model $F : \mathcal{G} \to \mathcal{Y}$ is trained for a classification task, an evasion attack is successful when the discretized prediction of the model changes to another class (in a targeted attack, this class is specified by the attack). Under a given *threat model* $T : \mathcal{G} \to \mathcal{P}(\mathcal{G})$, which specifies for each graph a set of lookalike graphs the adversary may construct, and a graph G to attack, the attack tries to find any $G' \in T(G)$ for which $\texttt{class}(F(G')) \neq \texttt{class}(F(G))$ (or in case of a targeted attack $\texttt{class}(F(G')) = c$ for some specified class c). We cannot directly apply gradient-based methods to find such a G' because the procedure `class` doesn't yield useful gradients. Instead, we can replace `class` by some differentiable target function $t : \mathcal{Y} \to \mathbb{R}$ that achieves high values only if the model output matches the adversary's goal (similar to a loss function, where high values indicate bad predictions) and optimize for $t \circ F$ via gradient-based methods. The impact of the target function goes beyond the

[1] Gradient-based poisoning attacks involve meta-gradients that track how a change in the graph structure affects the attack target after training on the changed attack. To compute such meta-gradients, we need to store all intermediate values for the backward pass, which requires GPU memory proportional to the number of epochs over which the meta-gradient is computed.

semantics of wrong predictions but also influences the convergence of gradient-based optimization techniques.

In summary, an attack on model $F : \mathcal{G} \to \mathcal{Y}$, graph $G \in \mathcal{G}$ with target function $t : \mathcal{Y} \to \mathbb{R}$ and threat model $T : \mathcal{G} \to \mathcal{P}(\mathcal{G})$ tries to solve the following optimization problem[2]:

$$\underset{G' \in T(G)}{\operatorname{argmax}} \; t(F(G')). \tag{4.1}$$

Target Functions

In a C-class classification setting, when the model output is a logit vector from $\mathcal{Y} = \mathbb{R}^C$, a sensible target function to switch predictions to class c is $t_c(\hat{y}) = \hat{y}_c - \sum_{i \neq c} \hat{y}_i$. For binary classification (when $\mathcal{Y} = \mathbb{R}$), we use the target function $t_y(\hat{y}) = (1 - 2y)\hat{y}$ where $y \in \{0, 1\}$ is the true class of a given graph and $\hat{y} \in \mathbb{R}$ the predicted logit. Note that a graph G with label y gets misclassified if and only if $t_y(F(G)) > 0$.

While there is a binary notion of a successful attack in a classification setting, there is no principled way to define such a notion in a regression setting because the predicted values have different meanings across datasets and domains. The loss function used for training the model indicates good predictions with low values and thus might be a suitable target function depending on the application. We train our regression models (where $\mathcal{Y} = \mathbb{R}$) with the L1-loss, which yields the target function $t(\hat{y}) = |\hat{y} - y|$. When applying a multistep gradient-based attack with this target function, the first few steps will implicitly decide whether to drive the predicted value $\hat{y}$ up or down. Since our models are highly non-linear, such an initial decision might not be optimal, which is why we try to individually increase and decrease the predicted value as much as possible to make our attack stronger. To this end, we use the target functions $t_\uparrow(\hat{y}) = \hat{y}$ and $t_\downarrow(\hat{y}) = -\hat{y}$.

4.2 Threat Models

From a security perspective, a *threat model* determines how much an attacker can control the inputs that are fed into a model, excluding drastic changes that can be easily discovered and discarded by a defender. The definition of a reasonable threat model heavily relies on the specific application domain and the abilities attributed to both attackers and defenders. Threat models are often parametrized by a *budget* $\varepsilon \in (0, \infty)$, which specifies the extent to which an attacker can change valid inputs without being detected by a defender. In a discrete setting, the simplest threat model is to count the number of changes an attacker is allowed to make compared to the initial input.

The **adjacency threat model**, which allows for up to ε adjacency changes, is

[2] In general, even this problem is hard to solve exactly due to the large size of $T(G)$, which grows exponentially in the number of allowed changes to G. Instead, our attacks aim to solve this problem approximately.

described by the function $T_a : \mathcal{G} \times \mathbb{N} \to \mathcal{P}(\mathcal{G})$, where

$$T_a(G, \varepsilon) = \{G' \mid \begin{aligned} & V = V' \wedge X = X' \\ & \wedge \forall e \in E \cap E'.Y(e) = Y'(e) \\ & \wedge |E \oplus E'| \leq \varepsilon\}, \end{aligned} \tag{4.2}$$

and $A \oplus B = (A \setminus B) \cup (B \setminus A)$ denotes the symmetric set difference. This threat model enforces that the perturbed graphs have the same nodes and node features, however, the edge features only need to agree on shared edges. In particular, newly introduced edges may have arbitrary features.

The **feature and adjacency threat model** $S_f : \mathcal{G} \times \mathbb{N} \to \mathcal{P}(\mathcal{G})$ builds upon the adjacency threat model and additionally allows for feature changes. Each value change in any discrete feature vector uses a budget of one. We denote the number of node feature changes by $\Delta X(G, G') = \sum_{v \in V \cap V'} \|X(v) - X'(v)\|_0$ and the number of edge feature changes by $\Delta Y(G, G') = \sum_{e \in E \cap E'} \|Y(e) - Y'(e)\|_0$, where $\|\cdot\|_0$ counts the number of non-zero elements. Formally, the threat model is

$$T_f(G, \varepsilon) = \{G' \mid V = V' \wedge |E \oplus E'| + \Delta X(G, G') + \Delta Y(G, G') \leq \varepsilon\}. \tag{4.3}$$

The goal of a threat model is to list all graphs where the defender cannot notice changes from a valid graph. In case a dataset contains graphs of very different sizes, whether changes are noticeable depend more on the number of changes relative to the graph size than the absolute number of changes. The previously described threat models are parametrized by an *absolute budget*, but can easily be adapted to a *relative budget*, which is used together with the graph size to derive an absolute budget, individualized to each graph. The relative budget that we use is relative to the number of edges M.

The threat models T_a and T_f are quite general in the sense that they allow for all kinds of changes. Depending on the domain, this might not be very realistic. In the molecular domain, for example, adding an edge corresponds to forming a bond between two atoms, which is likely not possible due to the fixed valency of the involved elements. The same problem arises when changing an edge feature that represents the type of bond. Node features for molecules might include the atom type, charge, or chirality and thus cannot be changed arbitrarily. In general, single changes to molecular graphs are likely to result in graphs that cannot be derived from valid and stable molecules. Threat models that are restricted to valid molecules are out of the scope of this thesis, as they require chemistry domain knowledge. A simple approach to address this limitation is to use the here presented attack to find potentially non-molecular adversarial examples and then to "project" them to the closest valid molecule. However, such a projection would almost certainly alter the total number of changes to the original molecule, resulting in a molecule that might land outside the budget restrictions imposed by our threat models. In line with the concept of unnoticeable changes, it could be more reasonable to consider threat models that aren't based on budgets that count the number of changes, but instead are based on how similar molecules are in a chemical sense, e.g. by comparing functional groups or the charges of atoms. We leave these problems and ideas for future research and focus on the threat models described above.

4.3 Random Perturbations and Simple Attacks

4.3.1 Random Perturbations

As a starting point, we wanted to see how sensitive the models are to random perturbations. To this end, we considered the following random changes in isolation (no combinations of changes): removing edges, adding edges (without introducing self-loops), "rewiring" edges, and changing node and edge features.

Adding Edges To remove k edges from a graph on M edges, we uniformly sampled a subset of k out of the M edges. The features of the newly added edges are sampled uniformly from the original graph, essentially by choosing an index into the edge feature tensor and copying the corresponding entry to the new feature.

Removing Edges To add k edges, we uniformly sampled a subset of size k from the $\binom{N}{2} - M$ missing edges. The accompanying edge features are sampled from the original edge features, where each feature vector is as likely as how often it appears in the original graph. This is a bit more restrictive than what our threat models allow, but it aligns with the attackers' goal of making the perturbations hard to detect by keeping the distribution of edge features similar.

Rewiring Edges "Rewiring" an edge means to removing an edge $\{v_i, v_j\}$ and introducing a new edge $\{v_i, v_k\}, i \neq k$. The rewired edges retain their original edge features. Although this change requires an adjacency budget of 2, it ensures that the degrees of at most two nodes (v_j and v_k) change, while arbitrary two adjacency changes may result in up to four node degree changes.

Changing Features To obtain a graph with k node features changes, we select k nodes uniformly at random. For each of these nodes, we construct a list of single change feature vectors and sample a new feature vector uniformly at random from this list. Edge feature changes are constructed similarly by selecting k edges uniformly at random and then changing one feature randomly per edge.

4.3.2 Brute Force

Since our graphs are rather small, it is viable to find the strongest attack for small budgets by trying all perturbations. For a budget of 1 and a graph on N nodes and M edges, there are M changes of removing existing nodes, $F_e \cdot \left(\binom{N}{2} - M\right)$ perturbations for adding an edge where the edge features of new edges may take on F_e values within a budget of 1. Additionally, there are $F_n \cdot N$ many possible 1-budget node feature changes and $F_e \cdot M$ many 1-budget edge feature changes, where F_n and F_e denote the total number of node and edge feature values. By trying all feature values (including the ones of the original graph), we need to evaluate the target function $M + F_e \cdot \binom{N}{2} + F_n \cdot N \leq \mathcal{O}(N^2)$ many times for a single graph to get the strongest attack of budget 1.

4.4 Adjacency Projected Gradient Descent

Our first gradient-based attack, AdjPGD, takes the approach outlined in [22] in attacking the edges. In summary, this attack relaxes the adjacency matrix $A \in \{0,1\}^{N\times N}$ to a tensor $\widetilde{A} \in [0,1]^{N\times N}$, which gets interpreted as a probability distribution over adjacency matrices where each edge $\{i,j\}$ independently occurs with probability $\widetilde{A}_{i,j}$. The algorithm then optimizes $\widetilde{A}$ to maximize a differentiable function $f : [0,1]^{N\times N} \to \mathbb{R}$ with projected gradient descent (PGD)[3]. The obtained distribution matrix is finally used to sample k adjacency matrices $A_1, \ldots, A_k \in \{0,1\}^{N\times N} \sim \text{Bernoulli}(\widetilde{A})$ (independent element-wise Bernoulli distribution) and returns $\operatorname{argmin}_{A_i \in \{A_1,\ldots,A_k\}} f(A_i)$. For a successful attack, the sampled matrices should (1) have a decent (constant) chance of being within budget constraints and (2) produce values that maximize f.

4.4.1 Algorithm

To achieve (1), the authors only consider distributions that meet the budget *in expectation*, hoping that this yields many adjacency matrices during sampling that do not violate the budget constraints. Given some distribution $\widetilde{A}$ and initial adjacency matrix, the expected budget used sampling from $\widetilde{A}$ is given by

$$B_A(\widetilde{A}) = \frac{1}{2}\sum_{i,j=1}^{N} |\widetilde{A} - A|_{i,j} \tag{4.4}$$

The correction factor $\frac{1}{2}$ is needed since we only consider undirected graphs.[4] Due to the same reason, we are only interested in symmetric distributions satisfying $\widetilde{A}^{\mathrm{T}} = \widetilde{A}$. In summary, the set of good distributions is

$$S_A = \left\{ \widetilde{A} \in [0,1]^{N\times N} \mid \widetilde{A}^{\mathrm{T}} = \widetilde{A} \wedge B_A(\widetilde{A}) \leq \varepsilon \right\}. \tag{4.5}$$

To obtain minimal values for the $f : [0,1]^{N\times N} \to \mathbb{R}$ (2), the authors apply projected gradient descent on the distribution matrix. Projected gradient descent (PGD) with base learning rate λ_0, budget ε and projection function $P_{S_A} : [0,1]^{N\times N} \to S_A$ and clamping function[5] $P_{[0,1]} : \mathbb{R}^{N\times N} \to [0,1]^{N\times N}$ performs the computations outlined in algorithm 1 on the next page.

The term $\frac{\varepsilon\lambda_0}{\sqrt{i}}$ provides a simple learning rate schedule[6] and the gradient term $\nabla_{\widetilde{A}} f(\widetilde{A}) + (\nabla_{\widetilde{A}} f(\widetilde{A}))^{\mathrm{T}}$ reflects that we operate on undirected graphs whose adjacency matrices can be parametrized by upper or lower triangular matrices. The "adjacency value" $a_{\{i,j\}}$ linking nodes i and j is present in both $\widetilde{A}_{i,j}$ and $\widetilde{A}_{j,i}$ ($a_{\{i,j\}} = \widetilde{A}_{i,j} =$

[3] We use this optimization algorithm to maximize a function, so projected gradient *ascent* might be more fitting in this scenario.

[4] The paper abstracts out perturbations on adjacency matrices to perturbations on vectors, whose elements may be embedded in adjacency matrices or other structures with binary features (presumably because it is more general and allows for a cleaner presentation). We have concretized the attack to adjacency matrices for undirected graphs.

[5] $P_{[0,1]}(X)_{i,j} = X_{i,j}$ if $X_{i,j} \in [0,1]$, $P_{[0,1]}(X)_{i,j} = 0$ if $X_{i,j} < 0$ and $P_{[0,1]}(X)_{i,j} = 1$ if $X_{i,j} > 1$.

[6] Which we have taken from the implementation of [13].

Algorithm 1 AdjPGD

1: $\widetilde{A} \leftarrow A$
2: **for** $i = 1 \ldots$ # PGD steps **do**
3: $\quad \lambda \leftarrow \frac{\lambda_0 \varepsilon}{\sqrt{i}}$
4: $\quad \widetilde{A} \leftarrow \widetilde{A} + \lambda D(\nabla_{\widetilde{A}} f(\widetilde{A}) + (\nabla_{\widetilde{A}} f(\widetilde{A}))^{\mathrm{T}})$
5: $\quad \widetilde{A} \leftarrow P_{S_A}(P_{[0,1]}(\widetilde{A}))$
6: **end for**

$\widetilde{A}_{j,i}$), meaning that

$$\frac{\partial f(\widetilde{A})}{\partial a_{\{i,j\}}} = \frac{\partial f(\widetilde{A})}{\partial \widetilde{A}_{i,j}} + \frac{\partial f(\widetilde{A})}{\partial \widetilde{A}_{j,i}},$$

which leads to the adjusted gradient expression. We set all diagonal gradient entries to 0 (indicated by applying $D(\cdot)$) to avoid self-loops and add the resulting gradient term to increase the value of f.

As a projection function, the authors suggest reducing the difference between a given $\widetilde{A} \in [0,1]^{N \times N}$ and the original adjacency matrix A uniformly among all entries by some $\mu \in [0,1]$ such that the distribution $P_{[0,1]}(\widetilde{A} + \mu(2A - \mathbf{1}))$, which is closer to A, perfectly fits into the budget (we use $\mathbf{1} \in \mathbb{R}^{N \times N}$ to denote the one matrix). Concretely, the projection function is

$$P_{S_A}(\widetilde{A}) = \begin{cases} \widetilde{A} & \text{if } B_A(\widetilde{A}) \leq \varepsilon \\ P_{[0,1]}(\widetilde{A} + \mu(2A - \mathbf{1})) & \text{if } B_A(P_{[0,1]}(\widetilde{A} + \mu(2A - \mathbf{1}))) = \varepsilon \end{cases} \tag{4.6}$$

They show that P_{S_A} minimizes the Frobenius norm to any element of S_A, i.e.

$$P_{S_A}(\widetilde{A}) = \operatorname{argmin}_{A' \in S_A} \|\widetilde{A} - A'\|_F^2 = \operatorname{argmin}_{A' \in S_A} \sum_{i,j=1}^{N} (\widetilde{A}_{i,j} - A'_{i,j})^2.$$

In order to implement P_{S_A}, we need to compute μ. This can be done via bisection, i.e. by halving the interval $[a, b]$ in which the true μ lies per step by checking if $\widetilde{A} + \mu(2A - \mathbf{1})$ exceeds the budget in expectation. Since $\mu \in [0,1]$, a constant number of steps can make the error on the true μ negligible.

Note that the algorithm doesn't explicitly enforce the invariance of $\widetilde{A}$'s symmetry. This is not necessary since $\widetilde{A}$ is initialized with a symmetric matrix (line 1), $X + X^{\mathrm{T}}$ is symmetric for any X and the difference of symmetric matrices is symmetric (line 3), and P_{S_A} has the same effect on every entry of $\widetilde{A}$ (line 4).

4.4.2 Attacking GNNs with features

To run the attack on our models, we need to choose a *differentiable* function $f : [0,1]^{N \times N} \to \mathbb{R}$ over which to optimize. Because sparsely-expressed MPNNs are not differentiable with respect to the input adjacency matrix, we construct differentiable surrogate models that compute a similar[7] function as the original models. Since the

[7] The surrogate model may deviate from the original model since we still evaluate the adversarial graphs on the original model. In case the original model admits obfuscated gradients with respect

adjacency also influences "how present" the edge features are, such a surrogate model $M : [0,1]^{N\times N} \times \mathbb{R}^{N\times N\times d_e} \to \mathcal{Y}$ is a function on both the adjacency matrix and dense edge features. With this, our f takes the form

$$f(A) = t\left(M(A, Y')\right), \tag{4.7}$$

where $t : \mathcal{Y} \to \mathbb{R}$ is a differentiable target function as introduced in section 4.1.

Y' is based on a completed edge feature tensor $\overline{Y} \in \mathbb{R}^{N\times N\times d_e}$, where given some edge features $Y : E \to \mathbb{R}^{d_e}$ we define $\overline{Y}_{i,j} = Y(\{i,j\})\ \forall \{i,j\} \in E$ and sample the missing edge features uniformly from the multiset of existing features, i.e.

$$\overline{Y}_{i,j} \overset{\text{iid}}{\sim} \mathcal{U}\left(\{\{Y(\{k,l\}) \mid \{k,l\} \in E\}\}\right)\ \forall \{i,j\} \notin E, i \neq j.$$

The diagonal entries $\overline{Y}_{i,i}$ are set to $\mathbf{0} \in \mathbb{R}^{d_e}$. Most of the dense surrogate models we construct later already limit the edge features with element-wise multiplication by the adjacency and thus compute the same function as their sparse counterparts when $Y' := \overline{Y}$. In particular, PPGN doesn't "limit" the dense representation of the edge features by the adjacency, which is why we manually do this element-wise multiplication with the adjacency matrix (i.e. $Y'_{i,j} = A_{i,j} \cdot \overline{Y}_{i,j}\ \forall i,j = 1,\dots,N$). In this case, the adjacency gradient $\nabla_A f(A)$ depends on the edge feature gradient $\nabla_{Y'} M(A, Y')$, so our surrogate models also need to be differentiable with respect to the edge features, which is given for all considered architectures. In general, multiplying the edge feature with the adjacency enables the adjacency gradient to correctly estimate the first-order impact of new edge features when flipping an edge.

In the following, we describe how we create differentiable surrogate models.

GIN(E)Conv Given a dense representation of node features $X \in \mathbb{R}^{N\times d_v}$, we can sum up the node features over the neighbors of all nodes by multiplying the adjacency matrix with X, since

$$\sum_{v_j \in \mathcal{N}(v_i)} X_{j,:} = A_{i,:} \cdot X = (A \cdot X)_{i,:}.$$

The GINConv operator adds the original features (scaled by $1+\varepsilon$) to the above sum. The new node representation is obtained by applying some MLP $h : \mathbb{R}^{d_e} \to \mathbb{R}^{d'_e}$. Since the MLP is evaluated independently for each node, we can simply evaluate the MLP h on each row of $(1+\varepsilon)X + A \cdot X$. The final dense version of the GINConv layer is thus

$$h\left((1+\varepsilon)X + A \cdot X\right). \tag{DenseGINConv}$$

This is how PyTorch Geometric implements the DenseGIN operator [24]. We adapt this approach to the GINEConv operator, which first combines the node features with the features of the combining edge before aggregating over the neighboring nodes. The main trick is to extend the node features from a tensor from $X \in \mathbb{R}^{N\times d_v}$ to $X' \in \mathbb{R}^{N\times N\times d_v}$ by repeating values along the new first index and then to consider $X' + Y$, which contains the vectors $\mathbf{x}_j^{(k-1)} + \mathbf{e}_{j,i}$ at entry i,j. Then GINEConv applies ReLU to each of these values and aggregates over the neighborhood. We

to its input, specially designed surrogate models computing a different function can improve the attack strength.

implement the aggregation over the neighborhood by first multiplying element-wise with the adjacency, denoted by $A \odot \mathrm{ReLU}(X' + Y)$, and then to sum along the first dimension, yielding a tensor in $\mathbb{R}^{N \times d_v}$. After the aggregates over the neighborhood are computed, GINEConv does the same computation as GINConv. The complete dense version is thus given by

$$h\left((1+\varepsilon)X + \sum_{2^{\text{nd}}\text{ index}} A \odot \mathrm{ReLU}(X' + Y)\right). \qquad \text{(DenseGINEConv)}$$

DropGIN(E) relies mostly on GIN(E)Conv operations, which can be made dense as described above. Dropping out nodes changes the computation graph by removing the impact of certain nodes for some runs (by setting the adjacency of incident edges to 0), but the surviving adjacency still propagates gradients, which means that replacing the GIN(E)Conv operators by dense counterparts suffices to obtain gradients for the adjacency.

PPGN is differentiable with respect to the adjacency and edge features by design as it operates on dense representations of adjacency and edge features.

SignNet is a GINE model with positional encodings. The GIN(E)Conv operators can be made differentiable as described above, which just leaves the positional encodings, which are constructed out of the eigenvectors of the graph Laplacian. PyTorch implements the function `torch.linalg.eigh` to compute the eigenvectors of a matrix, which can backpropagate gradients. A problem however is that the gradient computation is undefined if case of repeated eigenvalues and numerically unstable when there are close eigenvalues due to the internal computation of the term $\frac{1}{\min_{i \neq j} \lambda_i - \lambda_j}$. Real-world graphs such as molecules often have repeated eigenvalues of their Laplacian [10], which prevents the direct use of `torch.linalg.eigh`. We circumvent this problem by adding Gaussian noise[8] to the diagonal entries of the adjacency matrix in case `torch.linalg.eigh` doesn't compute a suitable gradient.

ESAN has two stages, (1) generating subgraphs and (2) applying GINEConv on various aggregates of these subgraphs. With the ND and ED policies, the first stage is differentiable for the same reasons as DropGIN(E). However, the better-performing EGO/EGO+ policies are not, with no clear alternative formulation that is differentiable. Instead of generating ego subgraphs in a way that is differentiable with respect to the adjacency matrix, we decided to construct the EGO graphs based on a discrete adjacency matrix, sampled from $\widetilde{A}$. To still get gradient information over the generated subgraphs, we multiply the discrete adjacency matrix of the ego subgraphs element-wise with the adjacency distribution $\widetilde{A}$ and use the result as the relaxed adjacency matrix for the subgraphs. The second stage can be made differentiable with respect to the subgraphs, again by replacing GIN(E)Conv operators with dense counterparts.

To improve the effectiveness of the attack, we experimented with replacing all ReLU activations with LeakyReLU activations that have slope $\alpha > 0$ for negative inputs, hoping that the additional gradient information over nodes with negative inputs speeds up convergence and enables the discovery of better distributions. However, experiments showed little difference between the effectiveness of ReLU and leaky ReLU with well-chosen learning rates, the results of which can be found in Figure

[8] First with a standard deviation of 0.001, and in case of failure with 0.01.

5.3. Because of this, we decided against using leaky ReLU instead of ReLU activations.

4.4.3 Hyperparameters

AdjPGD has hyperparameters that can influence the quality of the constructed adversarial example, namely the base learning rate λ_0, the number of PGD steps, and the number of samples. We fix the number of PGD steps and the number of samples since increasing them can only improve the attack and the selected values provided good results on all manually tested graphs in a reasonable time. The attack strength seemed to be quite sensitive to the base learning rate λ_0 in manual testing, so fine-tuning the learning rate based on a given budget, target function, and model is appropriate. To this end, we run the attack on a small and randomly selected subset of the data with various learning rates from a log-uniform grid and choose the value which gives the best target score. To reduce the number of attack evaluations, we implemented a two-stage search where the second stage covers a smaller space centered (in log space) around the best combination from the first stage.

4.5 Feature and Adjacency Projected Gradient Descent

Our feature and adjacency attack, AttrPGD, follows a similar approach as AdjPGD, where the graph structure and features are relaxed to distributions that are continually updated using gradient information and projected back such that the budget holds in expectation, and finally sample adversarial examples from the learned distributions. We represent a feature that can take k different values as a vector $a \in [0,1]^k$, where the i-th entry is the probability that the feature takes the i-th value during sampling. In case the feature initially has the j-th value, the initial distribution is $a = [\delta_{i,j}]_{i=1}^k$. Edge features for negative edges are initialized to a uniform distribution $\left[\frac{1}{k}\right]_{i=1}^k$. In the following, $X \in [0,1]^{N \times p_v}$ denotes the tensor representation of node feature distribution with p_v values and $Y \in \mathbb{R}^{N \times N \times p_e}$ the tensor of edge feature distributions.[9]

4.5.1 Algorithm

In contrast to the adjacency attack, this attack optimizes a function $f : [0,1]^{N \times N} \times [0,1]^{N \times p_v} \times [0,1]^{N \times N \times p_e} \to \mathbb{R}$ on an adjacency distribution $\widetilde{A} \in [0,1]^{N \times N}$, node feature distribution $\widetilde{X} \in [0,1]^{N \times p_v}$ and edge feature distribution $\widetilde{Y} \in [0,1]^{N \times N \times p_e}$. To do so, $\widetilde{A}, \widetilde{X}$ and $\widetilde{Y}$ are initialized with the respective initial distributions. Then,

[9] In practice, the nodes may have $d_v > 1$ discrete features and the edges $d_e > 1$, which complicates finding a suitable tensor representation when features have a different number of values. Given d features with $n_1, \ldots, n_d$ values respectively, we solve this problem in the implementation by embedding each feature into a distribution vector from $[0,1]^{n_k}$ and concatenating all these vectors to obtain a vector representation in $[0,1]^{n_1+\ldots+n_d}$ of all d features. As an example, the edge feature distributions are then represented as a tensor from $[0,1]^{N \times N \times (n_1+\ldots+n_d)}$. The individual feature distributions can simply be spliced out of this tensor. We omit these details to improve the presentation, although the presented formulas still apply and only require straightforward modifications.

over several steps, these distributions are updated with gradient information and projected back such that the budget holds in expectation.

Gradient Updates

As in AdjPDG, this attack follow the learning rate schedule $\lambda \leftarrow \frac{\lambda_0 \varepsilon}{\sqrt{i}}$ for step i given some initial learning rate λ_0 and absolute budget ε. The adjacency, node and edge feature distributions are then updated as follows:

$$\begin{aligned}
\widetilde{A}' &\leftarrow \widetilde{A} + \lambda D\left(\nabla_{\widetilde{A}} f(\widetilde{A}, \widetilde{X}, \widetilde{Y}) + \left(\nabla_{\widetilde{A}} f(\widetilde{A}, \widetilde{X}, \widetilde{Y})\right)^{\mathrm{T}}\right) \\
\widetilde{X}' &\leftarrow \widetilde{X} + \lambda \nabla_{\widetilde{X}} f(\widetilde{A}, \widetilde{X}, \widetilde{Y}) \\
\widetilde{Y}' &\leftarrow \widetilde{Y} + \lambda D\left(\nabla_{\widetilde{Y}} f(\widetilde{A}, \widetilde{X}, \widetilde{Y}) + \left(\nabla_{\widetilde{Y}} f(\widetilde{A}, \widetilde{X}, \widetilde{Y})\right)^{\mathrm{T}}\right).
\end{aligned}$$

The gradient expressions for $\widetilde{A}$ and $\widetilde{Y}$ include the transpose[10] of the gradient because we are attacking undirected graphs and $D(\cdot)$ sets diagonal entries to 0 to avoid self-loops. After computing all gradients on the old distributions, we simultaneously update all distributions:

$$\widetilde{A}, \widetilde{X}, \widetilde{Y} \leftarrow \widetilde{A}', \widetilde{X}', \widetilde{Y}'.$$

Expected Budget

After each update, the attack projects the distributions into the space

$$S_{A,X,Y} = \left\{(\widetilde{A}, \widetilde{X}, \widetilde{Y}) \mid \widetilde{A} = \widetilde{A}^{\mathrm{T}} \wedge \widetilde{Y} = \widetilde{Y}^{\mathrm{T}} \wedge B_A(\widetilde{A}) + B_X(\widetilde{X}) + B_Y(\widetilde{Y}) \leq \varepsilon\right\},$$

such that the budget ε holds in expectation.

The adjacency budget B_A is computed as in AdjPGD. The expected budget of a feature distribution $\widetilde{a} \in [0,1]^k$ given some initial distribution $a = [\delta_{i,j}]_{i=1}^k$ is $B_a(\widetilde{a}) = (\mathbf{1} - a) \cdot \widetilde{a}$. To compute the expected node feature budget we add up this expression over all nodes:

$$B_X(\widetilde{X}) = \sum_{i=1}^{N} (\mathbf{1} - X_i) \cdot \widetilde{X}_i. \tag{4.8}$$

The budget for edge features needs to be scaled by the respective adjacency value because we sample adjacency and features independently and there can only be a change in edge features if the corresponding edge is present. Per our feature and adjacency threat model (equation (4.3)), edge features for originally negative edges incur no budget. Therefore, the expected edge feature budget is given by

$$B_Y(\hat{Y}) = \sum_{i,j=1}^{N} A_{i,j} \widetilde{A}_{i,j} (\mathbf{1} - Y_{i,j}) \cdot \widetilde{Y}_{i,j}. \tag{4.9}$$

Note that the expression $(\mathbf{1} - Y_{i,j}) \cdot \widetilde{Y}_{i,j}$ only computes the correct expected budget if $Y_{i,j} = [\delta_{i,l}]_{i=1}^{p_e}$ for some $l \in \{1, \ldots, p_e\}$. This is not the case for negative edges, but they have no impact due to the multiplication with $A_{i,j} = 0$.

[10] The edge feature gradient is transposed along the first two dimensions.

Projection

As in AdjPGD, we clamp all adjacency values to the interval $[0, 1]$ before projecting into $S_{A,X,Y}$. The feature distributions are normalized after clamping to $[0, 1]$.

In case the triplet $(\widetilde{A}, \widetilde{X}, \widetilde{Y})$ exceeds the prescribed budget ε, the algorithm projects the triplet into $S_{A,X,Y}$ by uniformly decreasing the impact of each distribution by the smallest $\mu \in [0, 1]$ such that the budget holds in expectation (after clamping all values to the interval $[0, 1]$). As in AdjPGD, μ is determined using bisection.

Adjacency The adjacency budget is reduced as in AdjPGD, namely as

$$P_{[0,1]}(\widetilde{A} + \mu(2A - \mathbf{1})).$$

Node Features Given a single node attribute distribution $\widetilde{a} \in \mathbb{R}^{p_v}$ representing a feature that initially had the l-th value, we reduce its required budget using the following algorithm:

$$\widetilde{a} \leftarrow \left(1 - \frac{\mu}{\sum_{i \neq l} \widetilde{a}_i}\right) \widetilde{a} \qquad \text{(step 1)}$$

$$\widetilde{a} \leftarrow P_{[0,1]}(\widetilde{a}) \qquad \text{(step 2)}$$

$$\widetilde{a}_l \leftarrow 1 - \sum_{i \neq l} \widetilde{a}_i. \qquad \text{(step 3)}$$

We proportionally reduce each probability in step 1 and "fix" the probability of the initial value in step 3 to again obtain a valid distribution ($\sum_{i=1}^{p_v} \widetilde{a}_i = 1$). μ is scaled by the old expected budget $\sum_{i \neq l} \widetilde{a}_i$ to ensure that the new distribution requires no budget in case $\mu \geq \sum_{i \neq l} \widetilde{a}_i$, which mirrors the behavior of reducing the adjacency budget of a single adjacency value by μ. Step 2 ensures all values are positive in case $\mu > \sum_{i \neq l} \widetilde{a}_i$.

Edge Features Edge feature distributions are reduced similarly to node feature distributions. The only difference is in step 1, where the algorithm additionally scales the amount to reduce by the corresponding adjacency value $\widetilde{A}_{i,j}$ (before decreasing it by μ) and the initial adjacency value $A_{i,j} \in \{0, 1\}$:

$$\widetilde{a} \leftarrow \left(1 - A_{i,j}\widetilde{A}_{i,j}\frac{\mu}{\sum_{i \neq l} \widetilde{a}_i}\right) \widetilde{a}. \qquad \text{(step 1)}$$

The factor $A_{i,j}$ ensures that we only change distributions of initially present edges since edge features of negative edges incur no budget as specified by our threat model.

As a consequence of the factor $\widetilde{A}_{i,j}$, a reduction by μ doesn't result in a reduction of expected budget by μ when $\widetilde{A}_{i,j} < 1$ (even if $\mu \leq \sum_{i \neq l} \widetilde{a}$)[11]. We chose this behavior to prevent undoing learned feature distributions that only have little impact on the budget but might be important in later PGD steps or during sampling.

[11] Such a behavior would require a scaling by $\frac{1}{\widetilde{A}_{i,j}}$ instead of $\widetilde{A}_{i,j}$ since the expected budget is given by $\widetilde{A}_{i,j} \sum_{i \neq l} \widetilde{a}$.

Closing Remarks

Feature embeddings, as introduced in section 2.1, rely on discrete features, which are used to look up an embedding vector from a table $L : \{1, \ldots, p\} \to \mathbb{R}^d$. This procedure can be made differentiable with respect to a feature distribution $\widetilde{a} \in \mathbb{R}^d$ by embedding $\widetilde{a}$ as

$$L'(\widetilde{a}) = \sum_{i=1}^{p} \widetilde{a}_i L(i).$$

By adapting the feature embedding scheme to distributions, the surrogate models discussed in section 4.4.2 become differentiable with respect to the node and edge feature distributions.

Note that this algorithm can easily be adjusted to slightly different threat models, which e.g. weigh changes to the adjacency differently than changes to the features, or which have different budgets for the adjacency, node, and edge features.

4.6 Adversarial Training

Given some parametrized model f_θ and data distribution D, a supervised training procedure aims to find model parameters θ that minimize the expected loss, measured by the loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. This empirical risk minimization (ERM) framework is expressed as the following optimization problem:

$$\operatorname*{argmin}_\theta \mathbb{E}_{(x,y)\sim D} \left[l(f_\theta(x), y) \right]. \tag{4.10}$$

Adversarial training, introduced by [14] and popularized by [15], additionally aims to improve the robustness of the trained model, which under a threat model $T : \mathcal{G} \to \mathcal{P}(\mathcal{G})$ is inversely related to the worst loss value achieved among the threat model $\max_{G' \in T(G)} l(f_\theta(G'), y)$. To mitigate the effect of such adversarial perturbations, adversarial training solves the following saddle-point optimization problem instead:

$$\operatorname*{argmin}_\theta \rho(\theta) \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(G,y)\sim D} \left[\max_{G' \in T(G)} l(f_\theta(G'), y) \right]. \tag{4.11}$$

This optimization problem is usually solved by modifying the training procedure. In each gradient descent step, the model is trained on the pair

$$\left(\operatorname*{argmax}_{G' \in T(G)} l(f_\theta(G'), y),\ y \right)$$

instead of (G, y). The modified graph $\operatorname{argmax}_{G' \in T(G)} l(f_\theta(G'), y)$ is (approximately) found using an adversarial attack.

Chapter 5

Experiment and Results

This chapter describes the main experiment and provides data to answer the main question:

Are expressive GNNs more or less robust on graph-level tasks than less-expressive GNNs?

Regarding adversarial training and robust GNNs, we describe an additional experiment and analyze the resulting data to answer the following secondary questions:

How does the expressiveness of a model relate to robustness improvements due to adversarial training? Does the expressiveness of a model improve its performance on clean data when trained adversarially?

5.1 Methodology

There are multiple ways to define the robustness of machine learning models, even if we restrict ourselves to test-time perturbations of data.

The first is concerned with inputs that arise in practice, but are not specifically tailored to fool the model. This can include distribution shifts and natural variation in the data that wasn't observed during the training procedure. A natural way to evaluate this kind of robustness is to test the accuracy of models on data sampled from its underlying distribution. The problem with this approach is the combinatorial nature of graphs, which makes it difficult to specify the distribution of graph data. Instead, we probe for this kind of robustness using random perturbations in the experiments, as described in Section 4.3.

More commonly, robustness is defined via adversarial attacks. A threat model quantifies which perturbations an adversary may produce that are unnoticeable by the defender. Given a threat model, an adversary may aim to find the strongest change that is allowed, i.e. solve the optimization problem in equation (4.1) exactly. If the adversary has access to a model evaluation oracle, he may solve this problem exactly by brute-forcing all graphs in the threat model, which is possible since our threat models are finite. We test the robustness against an adversary with an evaluation oracle by running a brute force attack of budget 1.

Finally, we can take a practical security approach and only consider adversarial examples that can efficiently be found by an adversary. To get a lower bound on the security, we want to make as few assumptions about the adversary as possible, which is why we need to consider the strongest (efficient) adversary. For attacking machine learning models, research suggests that white-box[1] PGD-based optimization attacks are the strongest [13, 31]. We evaluate this robustness scenario using the AdjPGD and AttrPDG attacks described in Sections 4.4 and 4.5. We evaluate the strength of the PGD attack relative to the brute force attack (for budget 1) and the random sampling attack (for higher budgets).

In all scenarios, we consider two threat models: one in which the attacker is only allowed to change the adjacency (equation (4.2)) and one where the attacker may also change the node and edge features (equation (4.3)). We employ both absolute and relative budgets to test the models on different notions of unnoticeable changes.

5.2 Experimental Setup

The experiments are implemented in PyTorch [32] and PyTorch Geometric [24]. All architectures except for SignNet have an open-source implementation in PyTorch Geometric, which we adapted to our codebase for shared training, evaluation, and attack routines. The code is available on GitHub.

Datasets

We use the following datasets, which are commonly used to evaluate the performance of expressive GNNs.

Dataset	**Task**	**Metric**	**# Graphs per split**	N	M	d_v	d_e
MolHIV	binary classification	ROC-AUC	32'901/4'113/4'113	25.5	27.5	9	3
ZINC12k	regression	MAE	10'000/1'000/1'000	23.2	24.9	1	1
IMDB-BINARY	binary classification	accuracy	800/100/100	19.8	96.5	0	0
IMDB-MULTI	3-way classification	accuracy	1'200/150/150	13.0	65.9	0	0
MUTAG	binary classification	accuracy	150/19/19	17.9	19.8	7	4

Table 5.1: Dataset Overview: N is the average node count, M the average edge count, d_v the mean number of node features, and d_e the mean number of edge features.

MolHIV is part of the Open Graph Benchmark (OGB) [17] and includes 41'127 molecules with a binary classification task. The nodes (atoms) and edges (bonds) include rich features and are listed in table A.2. Some node features, such as the node degree or whether the atom is part of a ring, contain structural information which depends on the neighborhood of the atom. The class distribution of MolHIV is highly skewed - with a 96.5% / 3.5% class split, which is why MolHIV models are evaluated using a ROC-AUC score.

[1] Where the adversary has access to the model and its parameters.

ZINC12k contains 12'000 commercially available molecules, labeled by their penalized $\log P$ score[2] (regression task) [18, 19]. The penalized $\log P$ score is computed as $y = \log P + \texttt{SAS} - \texttt{cycles}$[3], where `cycles` is the number of cycles with at least six atoms. In contrast to the many features of MolHIV, ZINC12k only has a single feature per node, the atom number, and a single feature per edge, the bond type.

IMDB-BINARY and **IMDB-MULTI**, part of the TUDataset [21], contain 1'000 and 1'500 ego graphs with no node or edge features of actor collaborations. The task is to predict the film genre, of which IMDB-BINARY has 2 and IMDB-MULTI has 3. The classes are perfectly balanced (500/500 for IMDB-BINARY and 500/500/500 for IMDB-MULTI).

MUTAG is a small molecular dataset with just 188 graphs and also part of TUDataset [21]. The task is to predict one of two classes indicating the impact of the molecule on bacterial mutagenesis. Its small size makes it suitable for the adversarial training experiment, which is quite slow due to the additional overhead of re-computing adversarial examples in each epoch.

Models

We performed hyperparameter optimization for the model training with a parameter grid inspired by the OGB leaderboard and hyperparameters used in official implementations, specifically optimizing the base learning rate, batch size, and drop-out fraction. We selected the best hyperparameter combination for each architecture based on the validation score.

Finally, we trained five models on the best combination of hyperparameters with different seeds, which we use to evaluate the robustness of their architectures. All models were trained using the ADAM optimizer. We used the L1 loss for regression, the binary cross entropy loss for binary classification, and the cross entropy loss for multi-class classification. All models except SignNet were trained up to 300 epochs (1000 epochs for SignNet[4]) and stopping in case the validation score has not improved in the last 100 epochs (200 epochs for SignNet). The model was selected based on the best validation score achieved during training. We scheduled the learning rate with `ReduceLROnPlateau` that reduced the learning rate by a factor $\frac{1}{2}$ in case there has been no improvement in the best validation score in the last 25 epochs.

This model training and selection procedure differs from how TUDataset (IMDB-BINARY, IMDB-MULTI and MUTAG) models are usually evaluated. These models are usually trained on 10 folds (yielding a 9/1 train/test split each). Per epoch, the average performance on the test split over the 10 folds is computed and the best average is reported. However, the robustness evaluation requires a single model and some withheld test data. To achive this, we constructed a train, validation, and test split by dividing the training part of the first fold out of the 10 into a final train and validation split to obtain an 8/1/1 split. As with the other datasets, we train the models on the train split, perform model selection based on the validation split and evaluate the robustness of the selected models using the test split. ZINC12k and MolHIV come with a pre-determined train/validation/test split.

[2] Alternatively called constrained solubility.
[3] $\log P$ is the water-octanol partition coefficient and `SAS` the synthetic accessibility score.
[4] This follows the official implementation of SignNet for ZINC12k.

Since the IMDB datasets come without any node features, we use the node degrees as an index into a lookup table with 26 embedding vectors[5]. To attack this degree embedding scheme, we attach gradients to the embedding vector closest to the relaxed node degree, which is computed as the sum over the adjacency values for a given node.

In the following, we provide some implementation details on the architectures we used.

Baseline As a baseline for the random perturbations, we train models that only use the node features, but no adjacency information. These models apply the same 5 MLP layers to all node features, then combine the feature vectors via global mean pooling and further process the graph representation using a small MLP.

GINE Our GINE model consists of 5 layers composed of GINEConv, batchnorm, and ReLU. The outputs after every layer are independently mean-pooled to a graph representation and mapped to a value of the output shape. The output of the model is the sum of these per-layer outputs.

DropGINE The DropGINE architecture follows the GINE architecture described above closely. The same 5 layers are applied to each run independently. The runs are averaged together before mean-pooling after every layer. We set γ to the average node count, rounded to the nearest integer, do $r = \gamma$ runs, and drop out nodes with probability $p = \frac{2}{1+\gamma}$.

ESAN We test the DSS-GNN architecture with GINEConv layers as base encoders. The H-equivariant layer we use aggregates the subgraphs for the L^2 component by maxing the adjacencies (recovering the original adjacency) and averaging the node features. We exclusively used the EGO+ subgraph selection with 3 hops and sampling fraction 20% (i.e. using a randomly sampled subset of 20% of all subgraphs).

SignNet We compute sign-invariant positional encodings as $f(v_1, \dots, v_k) = \rho\left([\varphi(v_i) + \varphi(-v_i)]_{i=1}^{k}\right)$, where $\varphi : \mathbb{R}^{N\times k} \to \mathbb{R}^{N\times k}$ is the composition of multiple GIN layers and $\rho : \mathbb{R}^{N\times k} \to \mathbb{R}^{N\times k'}$ applies the same MPL independently to each of the N vectors in $\mathbb{R}^k$. We decided to use many eigenvectors to maximize expressiveness. For ZINC12k, we use $k = 37$, for MolHIV $k = 50$ and $k \in [15, 30]$ for the TUDataset models. The positional encodings are added to the existing node features and further processed using a usual GINE model. Since the SignNet model on seed 4 failed to converge on MolHIV with the selected hyperparameters, we replaced it with a model on seed 5.

PPGN We apply the PPGN architecture as outlined in section 3.4, with 5 blocks. The MLPs m_1, m_2 in each block consist of two linear layers with a leaky ReLU in between and the MLP m_3 is simply a linear layer. We refer to the code for the specifics of the used feature extractor $h : \mathbb{R}^{N\times N\times a} \to \mathbb{R}^{a'}$.

As mentioned previously, some official implementations of models use different edge embeddings in each layer, while the models we test only have one global embedding table. We tested the difference between GINE models with per-layer embeddings and global embeddings and have found little difference in the clean model performance across all datasets. Also, the official implementations of some models don't use edge features. In these cases, we replace the GINConv layers with GINEConv layers.

[5]Node degrees above 25 also map to the 26^{th} embedding vector.

Accuracy is the most important metric to evaluate attacks on classification models because it directly measures the percentage of inputs for which the attack is successful. Nonetheless, it doesn't report the confidence of a model in its predictions, which gives more insight into the effectiveness of attacks that only achieve a few mispredictions than the accuracy. Therefore we also report the average *label difference* (lbl. diff.) for classification models, which we define as $|\sigma(\hat{y}) - y|$ for binary classification tasks with label $y \in \{0, 1\}$, predicted logit $\hat{y} \in \mathbb{R}$ and the sigmoid activation function $\sigma : \mathbb{R} \to [0, 1]$. For C-class classification tasks, we define the label difference as $\sum_{i=1}^{C} |\text{softmax}(\hat{y})_i - \mathbb{1}_{y=i}|$ for the logit vector $\hat{y} \in \mathbb{R}^C$ and the label $y \in \{1, \ldots, C\}$.

Model Type	**MolHIV**			**ZINC12k**	
	ROC-AUC ↑	accuracy ↑	label difference ↓	MAE ↓	normalized MAE ↓
Baseline	0.722 ± 0.012	0.967 ± 0.002	0.066 ± 0.004	0.692 ± 0.002	0.343 ± 0.001
meanGINE	0.770 ± 0.005	0.968 ± 0.001	0.050 ± 0.004	0.363 ± 0.011	0.180 ± 0.005
GIN	0.749 ± 0.007	0.967 ± 0.003	0.054 ± 0.009	0.299 ± 0.013	0.148 ± 0.007
GINE	0.757 ± 0.035	0.965 ± 0.002	0.072 ± 0.023	0.246 ± 0.008	0.122 ± 0.004
SignNet	0.723 ± 0.019	0.965 ± 0.003	0.057 ± 0.003	0.176 ± 0.035	0.087 ± 0.017
DropGINE	0.771 ± 0.009	0.967 ± 0.001	0.051 ± 0.002	0.251 ± 0.005	0.124 ± 0.003
ESAN	0.727 ± 0.020	0.964 ± 0.006	0.068 ± 0.020	0.184 ± 0.018	0.091 ± 0.009
PPGN	0.756 ± 0.008	0.968 ± 0.001	0.052 ± 0.009	0.147 ± 0.003	0.073 ± 0.001

Model Type	**IMDB-BINARY**		**IMDB-MULTI**	
	accuracy ↑	label difference ↓	accuracy ↑	label difference ↓
Baseline	0.632 ± 0.016	0.361 ± 0.012	0.481 ± 0.015	0.417 ± 0.001
meanGINE	0.678 ± 0.030	0.343 ± 0.021	0.493 ± 0.008	0.411 ± 0.001
GINE	0.684 ± 0.036	0.333 ± 0.031	0.487 ± 0.016	0.412 ± 0.002
SignNet	0.698 ± 0.015	0.402 ± 0.014	0.389 ± 0.032	0.435 ± 0.004
DropGINE	0.720 ± 0.037	0.308 ± 0.027	0.485 ± 0.007	0.412 ± 0.001
ESAN	0.708 ± 0.038	0.308 ± 0.023	0.464 ± 0.017	0.417 ± 0.003
PPGN	0.672 ± 0.018	0.335 ± 0.015	0.451 ± 0.010	0.422 ± 0.002

Table 5.2: **Model Metrics on Clean Data** (↑: higher is better, ↓: lower is better)

Attacks

We attacked the binary classification datasets with the target function $t_y(\hat{y}) = (1 - 2y)\hat{y}$, which increases the logit output $\hat{y} \in \mathbb{R}$ in case $y = 0$ and decreases it when $y = 1$. To improve the strength of the attacks on the regression dataset ZINC12k, we individually try to increase and decrease the value in separate attacks using the target functions $t_\uparrow(\hat{y}) = \hat{y}$ and $t_\downarrow(\hat{y}) = -\hat{y}$ and only report the stronger attack. For IMDB-MULTI, which is a multi-class classification dataset, we use the logit margin $l_y(\hat{y}) = \max_{i \neq y} \hat{y}_i - \hat{y}_y$ for the output logits $\hat{y} \in \mathbb{R}^C$ and the original class $y \in \{1, \ldots, C\}$. These choices are inspired by [13], which recommends using the logit margin instead of the original model loss as a target function.

All adversarial graphs are evaluated on the original models. For the gradient-based attacks, we convert the sparse data to a dense representation for PGD to obtain a distribution over input graphs, and then evaluate the sampled graphs in a sparse representation on the original models.

Feature Changes For most perturbations that can change features, we allowed the perturbed feature to take on any value that occurs at least once in the dataset

(although the combination with other values in the same feature vector may be new). The reason for excluding values that never occur is that the associated embedding vectors were not trained and thus are random. From a security perspective, never seen feature values could easily be detected, so they should be excluded. Exceptions are the node features of MolHIV for the random perturbations and brute force attacks, which we restricted further to only include changes to the "atom number" feature, and only to atoms that occur in at least 0.1% of features. The reason for this is to give the random perturbations a reasonable chance to find an impactful change, as early experiments suggested that the atomic numbers of frequent atoms have the largest effect. Another benefit is that this restriction significantly speeds up the brute force search over MolHIV's many features.

ZINC12k & Normalized MAE Instead of the MAE, we report the normalized MAE which is obtained by re-scaling the model predictions and labels by the test data mean $\mu = 0.012$ and standard deviation $\sigma = 2.017$, and then computing the MAE between the adjusted predictions and labels. Some models on ZINC12k achieve extreme deviations from their initial prediction on a few graphs, which have a strong impact on the MAE. We decided to limit the impact of these outliers by removing 5% of graphs per model and budget which achieve the highest deviation compared to the label from all reported data. Also, one of the SignNet models produced extreme values even after discarding 5% of graphs (e.g. with a 255.855 normalized MAE under an adjacency brute force attack), so we replaced it with a model with a different seed.

Since the label of ZINC12k molecules depends on the number of cycles in a graph, which differs in attacked graphs, the models might correctly adjust their output to account for a change in the number of cycles. To not punish such behavior, we additionally evaluated the (normalized) MAE of the generated adversarial examples with a cycle-corrected label.

AUC-ROC Score For the active attacks on classification models we don't report AUC-ROC scores since this metric relies on how the predictions across the dataset relate instead of their values alone. On a binary classification task with balanced classes and a fixed threshold, it is possible to mispredict half of the data but still have a perfect ROC score.

5.3 Robustness under Brute Force Attack

The results of the brute force attacks are presented below. For ZINC12k, we include scores for the original as well as the cycle-corrected label. For both, we individually select the strongest change amongst all perturbation types and targets and discard the 5% of graphs with the strongest absolute error.

Additionally, we studied the distribution of perturbation types that lead to the strongest change.

Model Type	**MolHIV**			
	accuracy ↑	Δ accuracy ↓	label difference ↓	Δ label difference ↓
meanGINE	0.947 ± 0.010	0.021 ± 0.010	0.099 ± 0.020	0.049 ± 0.016
GIN	0.888 ± 0.079	0.078 ± 0.077	0.175 ± 0.077	0.120 ± 0.070
GINE	0.868 ± 0.056	0.097 ± 0.056	0.209 ± 0.056	0.137 ± 0.055
SignNet	0.475 ± 0.123	0.489 ± 0.121	0.506 ± 0.101	0.450 ± 0.101
DropGINE	0.884 ± 0.036	0.083 ± 0.036	0.175 ± 0.032	0.124 ± 0.031
ESAN	0.752 ± 0.172	0.212 ± 0.167	0.307 ± 0.137	0.239 ± 0.121
PPGN	$\mathbf{0.961 \pm 0.006}$	$\mathbf{0.008 \pm 0.006}$	$\mathbf{0.078 \pm 0.012}$	$\mathbf{0.026 \pm 0.008}$

Model Type	**ZINC12k**			
	original label		cycle-corrected label	
	norm. MAE ↓	Δ norm. MAE ↓	norm. MAE ↓	Δ norm. MAE ↓
meanGINE	$\mathbf{1.139 \pm 0.132}$	$\mathbf{0.959 \pm 0.134}$	$\mathbf{1.832 \pm 0.070}$	$\mathbf{1.652 \pm 0.069}$
GIN	2.760 ± 0.116	2.612 ± 0.119	2.874 ± 0.108	2.726 ± 0.112
GINE	1.529 ± 0.118	1.407 ± 0.118	2.027 ± 0.076	1.905 ± 0.075
SignNet	2.144 ± 0.487	2.056 ± 0.501	$\mathbf{1.869 \pm 0.270}$	1.782 ± 0.269
DropGINE	1.487 ± 0.082	1.363 ± 0.084	2.055 ± 0.057	1.931 ± 0.060
ESAN	1.862 ± 0.445	1.770 ± 0.450	1.947 ± 0.366	1.855 ± 0.370
PPGN	1.721 ± 0.200	1.648 ± 0.202	$\mathbf{1.729 \pm 0.160}$	$\mathbf{1.656 \pm 0.162}$

Model Type	**IMDB-BINARY**			
	accuracy ↑	Δ accuracy ↓	label difference ↓	Δ label difference ↓
meanGINE	0.456 ± 0.043	0.222 ± 0.056	0.552 ± 0.036	0.209 ± 0.056
GINE	0.496 ± 0.032	0.188 ± 0.065	$\mathbf{0.495 \pm 0.033}$	0.162 ± 0.060
SignNet	$\mathbf{0.598 \pm 0.060}$	$\mathbf{0.100 \pm 0.060}$	$\mathbf{0.459 \pm 0.043}$	$\mathbf{0.057 \pm 0.045}$
DropGINE	0.434 ± 0.062	0.286 ± 0.089	0.540 ± 0.057	0.231 ± 0.082
ESAN	0.458 ± 0.026	0.250 ± 0.021	0.516 ± 0.023	0.208 ± 0.031
PPGN	0.502 ± 0.030	0.170 ± 0.041	$\mathbf{0.473 \pm 0.009}$	0.138 ± 0.022

Model Type	**IMDB-MULTI**			
	accuracy ↑	Δ accuracy ↓	label difference ↓	Δ label difference ↓
meanGINE	0.291 ± 0.080	0.203 ± 0.085	0.451 ± 0.014	0.041 ± 0.014
GINE	0.353 ± 0.045	0.133 ± 0.056	0.443 ± 0.009	0.032 ± 0.009
SignNet	0.320 ± 0.052	0.069 ± 0.036	0.445 ± 0.007	$\mathbf{0.010 \pm 0.006}$
DropGINE	0.303 ± 0.104	0.183 ± 0.099	0.452 ± 0.013	0.040 ± 0.012
ESAN	0.316 ± 0.080	0.148 ± 0.074	0.446 ± 0.010	0.029 ± 0.009
PPGN	$\mathbf{0.396 \pm 0.008}$	$\mathbf{0.055 \pm 0.007}$	$\mathbf{0.436 \pm 0.004}$	$\mathbf{0.014 \pm 0.003}$

Table 5.3: **Adjacency Bruteforce Results** (↑: higher is better, ↓: lower is better)

Model Type	MolHIV			
	accuracy ↑	Δ accuracy ↓	label difference ↓	Δ label difference ↓
meanGINE	0.567 ± 0.288	0.401 ± 0.288	0.460 ± 0.240	0.411 ± 0.236
GINE	0.587 ± 0.209	0.378 ± 0.208	0.444 ± 0.165	0.372 ± 0.142
SignNet	0.468 ± 0.121	0.497 ± 0.119	0.511 ± 0.099	0.455 ± 0.100
DropGINE	0.732 ± 0.080	0.235 ± 0.081	0.324 ± 0.061	0.274 ± 0.061
ESAN	0.740 ± 0.178	0.224 ± 0.173	0.321 ± 0.141	0.254 ± 0.125
PPGN	$\mathbf{0.929 \pm 0.025}$	$\mathbf{0.039 \pm 0.025}$	$\mathbf{0.135 \pm 0.023}$	$\mathbf{0.083 \pm 0.024}$

Model Type	ZINC12k			
	original label		cycle-corrected label	
	norm. MAE ↓	Δ norm. MAE ↓	norm. MAE ↓	Δ norm. MAE ↓
meanGINE	2.477 ± 0.528	2.297 ± 0.532	2.654 ± 0.512	2.474 ± 0.517
GINE	2.325 ± 0.203	2.203 ± 0.201	2.490 ± 0.197	2.368 ± 0.195
SignNet	2.418 ± 0.623	2.330 ± 0.636	2.285 ± 0.531	2.198 ± 0.540
DropGINE	$\mathbf{2.156 \pm 0.144}$	$\mathbf{2.031 \pm 0.147}$	2.343 ± 0.112	2.218 ± 0.115
ESAN	2.323 ± 0.309	2.232 ± 0.314	2.276 ± 0.252	2.184 ± 0.256
PPGN	2.311 ± 0.286	2.238 ± 0.287	2.249 ± 0.315	2.176 ± 0.315

Table 5.4: **Attribute and Adjacency Bruteforce Results** (↑: higher is better, ↓: lower is better)

Perturbation	meanGINE	GIN	GINE	SignNet	DropGINE	ESAN	PPGN
added edge	73.1%	93.3%	93.7%	82.7%	92.6%	82.8%	93.1%
dropped edge	26.9%	6.7%	6.3%	17.3%	7.4%	17.2%	6.9%

Perturbation	meanGINE	GINE	SignNet	DropGINE	ESAN	PPGN
adjacency	2.5%	18.8%	95.5%	19.4%	79.6%	10.8%
node attr.	97.4%	81.1%	0.0%	80.5%	6.7%	85.7%
edge attr.	0.2%	0.1%	4.5%	0.1%	13.6%	3.5%

Table 5.5: Distribution of perturbation types amongst bruteforce attack on MolHIV

Observations and Discussion

Adjacency Bruteforce

MolHIV PPGN is the most robust architecture on MolHIV, where it loses almost no accuracy and keeps a low label difference under a single adjacency change. The next closest architecture, meanGINE, also achieves very good scores. SignNet loses by far the most accuracy under the strongest single adjacency change.

ZINC12k For ZINC12k with cycle-corrected labels, meanGINE, SignNet, and PPGN are the most robust architectures, but meanGINE beats all other models by quite a margin with the original label. However, with normalized MAEs of above 1, none of the models are robust enough to be reasonably used under brute force attacks. Interestingly, the only model that benefits from the cycle-correction of labels is SignNet, which could be explained by the ability of BasisNet to count the number of small cycles[6]. However, the score of the GIN and ESAN models also seems

[6]Each added or removed cycle changes the normalized MAE by $\frac{1}{\sigma} = 0.496$. Considering that not all adjacency changes alter the number of cycles with at least 6 nodes, this might reasonably explain a difference of about 0.275.

to not be impacted much, and the GIN models have no apparent advantage in terms of robustness.

IMDB Although SignNet performs badly on MolHIV, it is the most robust on IMDB-BINARY, with GINE and PPGN not far off. On IMDB-MULTI, PPGN beats all the other architectures in terms of accuracy but is followed closely by all other architectures in terms of label difference[7].

Attribute and Adjacency Bruteforce

MolHIV Compared to the adjacency brute force attack, the low-expressive models meanGINE and GINE suffer most from adversarial attribute changes on MolHIV. PPGN is the most robust architecture in this scenario, performing similarly with unperturbed inputs ever under this attack.

ZINC12k On ZINC12k, all models perform similarly. With the original label, DropGINE has an advantage. The high deviations between models of the same architecture, when evaluated with cycle-corrected labels, make it hard to rank the models but meanGINE performs best on average.

These results suggest that PPGN, which is the most expressive architecture tested, is also the most robust under adjacency and attribute brute force attacks across the tested datasets. However, SignNet performs well on IMDB-BINARY and meanGINE behaves similarly to PPGN in both MolHIV and ZINC12k under the adjacency brute force attack. It is not surprising that the variation across architectures is lower for the attribute and adjacency brute force attacks compared to just the adjacency brute force attacks since the more expressive GNNs are only more expressive for the graph structure, not the features.

Regarding the distribution of perturbations, which are presented in tables 5.5 and A.4, it appears that adding an edge can have more impact than removing an edge. This doesn't seem to be purely a consequence of the attribute threat model (which allows arbitrary features for new edges) since the GIN models behave similarly.

[7] This difference is likely due to the binary cross entropy loss that was used to train the models, which tends to optimize for accuracy instead of label difference.

5.4 Robustness under Random Perturbations

We sample 5 different graphs for each of the perturbation types and budgets. The results are separated into adjacency changes (average over the values obtained from adding and removing edges), attribute change (average over the node and edge feature modifications), and edge rewirings. Detailed plots per model type and budget can be found in the appendix (Fig. A.1 and A.2 for adjacency perturbations, Fig. A.3 for attribute perturbations, and A.4 for rewirings). The models were evaluated on the following absolute and relative budgets:

$$\begin{aligned} \varepsilon &\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40\} \\ \varepsilon/M &\in \{1\%, 2\%, 3\%, 4\%, 5\%, 6\%, 7\%, 8\%, 9\%, 10\%, 12\%, 14\%, 16\%, 18\%, 20\%, 30\%, 40\%, 50\%\}\,. \end{aligned}$$

To combine the results across budgets into a single score per model, we define the *Cumulative Relative Robustness* (CRR) score, which is inspired by the Area-Under-Curve score introduced in [13] and compares scores $x_1, \ldots, x_n$ to a baseline value b using a scaled sum of per-budget above-baseline values $\max\{0, x_i - b\}$[8]. For a sequence $\{(\varepsilon_i, x_i)\}_{i=1}^n$ of per-budget scores x_i sorted by budget $\varepsilon_1 < \ldots < \varepsilon_n$, the CRR score is computed as

$$\mathrm{CRR}\left(\{(\varepsilon_i, x_i)\}_{i=1}^n\right) = \frac{1}{\varepsilon_n - \varepsilon_0}\left(\sum_{i=1}^{n}(\varepsilon_i - \varepsilon_{i-1})\max\{0, x_i - b\}\right), \tag{5.1}$$

where $\varepsilon_0 = \varepsilon_1 - (\varepsilon_2 - \varepsilon_1)$ is chosen such that x_1 has the same impact as x_2. Models achieve high CRR scores if they initially perform better than the baseline and continue to beat the baseline, even under increasing budgets. The baseline value we use is the average clean performance of five MLP baseline models.

The results for the MolHIV, ZINC12k, and IMDB-BINARY datasets are presented in the tables below. We do not show results for IMDB-MULTI since the baseline values for both accuracy and label difference are such that most architectures already perform worse for budgets in the range 0 to 3 (see figures A.1, A.2, A.3, and A.4). In such cases, the CRR score is mostly determined by the clean performance instead of the performance under perturbations. For the same reason, we omit accuracy CRR scores for MolHIV and CRR scores with cycle-corrected labels for ZINC12k.

Observations and Discussion

Adjacency Perturbations

MolHIV The MolHIV CRR scores have very high variation, which can be attributed to the strong MolHIV baselines (96.7% accuracy, 0.066 label difference). The per-budget plots in Fig. A.1 and A.2 reveal that especially PPGN is the most robust under random adjacency perturbations with high absolute and relative budgets, both in terms of accuracy and label difference. Under increasing budgets, the performance of some models (GIN, SignNet on rel. budget) initially decreases but eventually improves again. This might be a result of the class skew of MolHIV, which incentivizes models to predict the more frequent class given an unknown input.

[8] In case a lower score is better, we use $\max\{0, b - x_i\}$ instead.

Model Type	**MolHIV** - label difference CRR		**ZINC12k** - normalized MAE CRR	
	abs. budget	rel. budget	abs. budget	rel. budget
meanGINE	0.515 ± 0.640	$\mathbf{0.872 \pm 0.453}$	0.959 ± 0.085	1.253 ± 0.139
GIN	$\mathbf{1.449 \pm 0.897}$	0.178 ± 0.145	0.703 ± 0.056	0.507 ± 0.068
GINE	0.019 ± 0.026	0.033 ± 0.046	1.050 ± 0.043	1.076 ± 0.107
SignNet	$\mathbf{1.095 \pm 1.472}$	$\mathbf{0.701 \pm 0.955}$	0.969 ± 0.143	0.737 ± 0.213
DropGINE	0.262 ± 0.186	$\mathbf{0.482 \pm 0.206}$	0.971 ± 0.056	0.912 ± 0.115
ESAN	0.037 ± 0.051	0.084 ± 0.116	1.102 ± 0.053	1.001 ± 0.055
PPGN	$\mathbf{0.699 \pm 0.496}$	$\mathbf{1.163 \pm 0.686}$	$\mathbf{1.288 \pm 0.023}$	$\mathbf{1.368 \pm 0.066}$

Model Type	**IMDB-BINARY**			
	absolute budget		**relative budget**	
	accuracy CRR	lbl. diff. CRR	accuracy CRR	lbl. diff. CRR
meanGINE	0.245 ± 0.185	0.074 ± 0.106	0.264 ± 0.204	0.075 ± 0.094
GINE	0.472 ± 0.183	0.159 ± 0.175	0.535 ± 0.191	0.186 ± 0.194
SignNet	$\mathbf{0.765 \pm 0.247}$	0.000 ± 0.000	$\mathbf{0.973 \pm 0.190}$	0.000 ± 0.000
DropGINE	0.498 ± 0.062	0.254 ± 0.122	0.514 ± 0.151	0.236 ± 0.084
ESAN	$\mathbf{0.530 \pm 0.297}$	$\mathbf{0.263 \pm 0.135}$	0.548 ± 0.305	0.244 ± 0.135
PPGN	0.363 ± 0.123	0.129 ± 0.093	0.492 ± 0.229	0.149 ± 0.139

Table 5.6: CRR scores for **random adjacency changes** (higher is better)

Model Type	**MolHIV** - label difference CRR		**ZINC12k** - normalized MAE CRR	
	abs. budget	rel. budget	abs. budget	rel. budget
meanGINE	1.029 ± 1.050	1.289 ± 0.841	0.526 ± 0.018	0.320 ± 0.011
GIN	1.125 ± 0.777	0.848 ± 0.758	0.629 ± 0.021	0.382 ± 0.013
GINE	0.863 ± 0.914	0.565 ± 0.720	0.766 ± 0.044	0.511 ± 0.056
SignNet	0.362 ± 0.392	0.339 ± 0.261	1.000 ± 0.061	$\mathbf{0.798 \pm 0.077}$
DropGINE	1.173 ± 0.641	1.216 ± 0.185	0.763 ± 0.055	0.522 ± 0.086
ESAN	0.325 ± 0.611	0.400 ± 0.645	0.912 ± 0.061	0.659 ± 0.090
PPGN	0.580 ± 0.426	0.940 ± 0.592	$\mathbf{1.050 \pm 0.029}$	$\mathbf{0.834 \pm 0.052}$

Table 5.7: CRR scores for **random attribute changes** (higher is better)

ZINC12k In terms of CRR score, PPGN beats the other models. The per-budget plots (Fig. A.1) show this score is largely influenced by the scores on budgets 0 and 1, and that PPGN as well as SignNet are significantly less stable than all other architectures with higher budgets. For PPGN, this instability is likely explained by the repeated matrix multiplication of the input adjacency matrix, and for SignNet it is likely a result of the instability of the Laplacian eigenvectors[9].

IMDB-BINARY SignNet achieves the highest CRR scores for accuracy, but has CRR scores of 0 for the label difference because it fails to beat the baseline even under no perturbations. On average ESAN achieves the highest CRR scores for the label difference.

Attribute Perturbations

MolHIV As with the adjacency perturbations RCC scores, the attribute RCC scores for MolHIV have very high standard deviations, which makes a comparison difficult.

[9]Also note that the used SignNet model has many layers, 8 for the Eigenvector embedding followed by 16 GINE layers.

However, meanGINE and DropGINE perform best on average.

ZINC12k As with the adjacency perturbation results, the CRR scores are mostly influenced by budgets 0 and 1, which again makes heavily biases the CRR scores towards the clean data performance of the models. Because of this, PPGN beats the other models in terms of CRR but again seems unstable for higher budgets.

5.5 Robustness under Gradient-Based Attacks

We apply AdjPDG and AttrPGD as introduced in Chapter 4. Because AdjPGD randomly samples the features of missing edges, we repeat the attack 3 times and only report the strongest change.[10] The number of gradient update steps is fixed to 25 and 250 graphs are sampled, amongst which only the strongest perturbation is recorded. When reducing the expected budget, we need to compute μ using bisection. We use 18 steps, which drives the error on the true μ to at most 2^{-18}.

Due to time constraints, we were not able to complete the PGD attacks on MolHIV for relative budgets.

Observations and Discussion

MolHIV On both AdjPGD and AttrPGD, PPGN outperforms all other models by quite a margin. For AdjPGD, PPGN is followed by meanGINE, and finally a cluster of all other models. meanGINE performs worse on AttrPGD and falls within a cluster of architectures that are superseded by PPGN and ESAN.

ZINC12k On AdjPGD, both PPGN and SignNet perform well on a budget of 1, but their MAE seems to grow super-exponentially for absolute budgets of 2 to 5, which is similar to their unstable behavior under random adjacency perturbations. SignNet and PPGN also perform well with relative budgets, as do GINE, DropGINE, and meanGINE. Excluding the instabilities of PPGN and SignNet on higher budgets, ESAN and GIN seem to be the least robust under an AdjPGD attack.

Under an AttrPGD attack, all models seem to perform similarly. The exceptions are again SignNet and PPGN on absolute budgets above 3, and PPGN achieves less error than the group of all other architectures across relative budgets.

[10] We briefly experimented with an attack that optimizes for the adjacency and attributes of missing edges (essentially by zeroing the gradients for node features and edge features of positive edges in AttrPGD) but decided against using this attack because it performed very similarly to AdjPGD and was a bit slower. We validate the attack strength of AdjPGD by comparison with brute force attacks later, where we will also note that changes to the node features and adjacency tend to allow for higher changes as edge features.

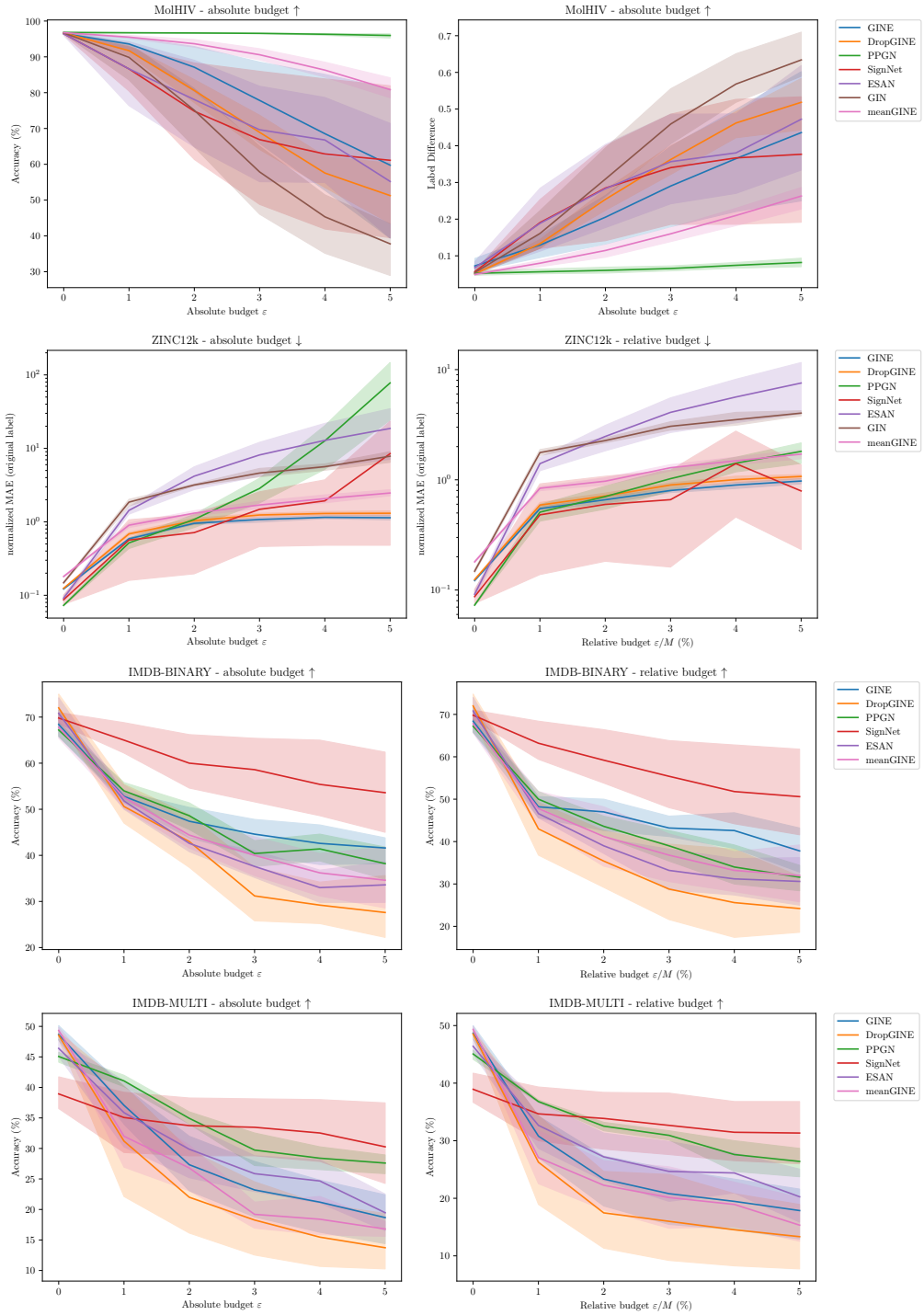


Figure 5.1: **AdjPGD Results** (↑: higher is better, ↓: lower is better)

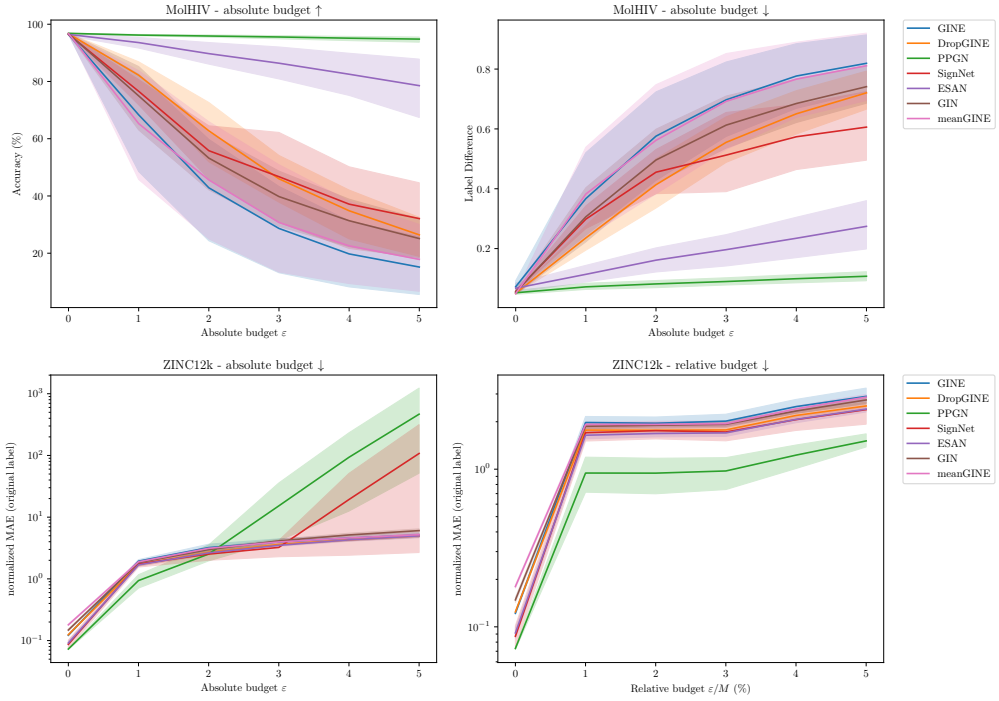


Figure 5.2: **AttrPGD Results** (↑: higher is better, ↓: lower is better)

5.5.1 Attack Strength

We briefly present some results to evaluate the attack strength of AdjPGD and AttrPGD.

Hyperparameter Tuning

As mentioned in Section 4.4.2, we briefly experimented with leaky ReLU activation functions[11] for surrogate models in hopes that additional gradient information flowing over nodes with negative inputs yields a stronger PGD attack. To evaluate the effectiveness of this method, we tracked the value of the target function on surrogate models with ReLU activation functions to get a clean target value. To find suitable learning rates, we performed a hyperparameter search over both learning rate λ_0 and negative slope α and used the learning rate from the best combination. The data for two such experiments are presented below in Fig. 5.3.

It turned out that the negative slope has little impact on the achieved target value, which is why we abandoned this idea. However, these results indicate that 25 PGD steps are sufficient, with significantly less improvement in the target value after 10 steps.

[11] The leaky ReLU activation function output αx for negative inputs x, instead of 0 as is the case with ReLU. The negative slope α is a small constant.

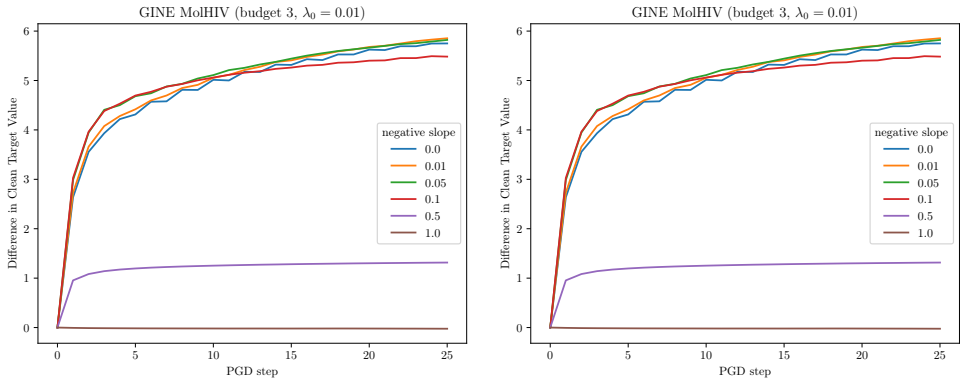


Figure 5.3: Negative Slope (higher is better)

Comparison to Bruteforce

To evaluate the performance of the PGD-based attacks for low budgets, we compare them to the brute-force attacks. As metrics, we present the relative MAEs for regression tasks, the gap between label differences for classification tasks, and how often the PGD-based attacks find a target value that matches the value found by the brute force attacks. For DropGINE and ESAN, which are not deterministic, we also count a match if PGD finds a strictly stronger attack.

The table below presents these values for MolHIV, and table A.5 for ZINC12k. Overall, the gradient-based attacks perform reasonably well relative to the brute force attacks.

AdjPGD - MolHIV

Model Type	Bruteforce - AdjPGD lbl. diff.	% of matched lbl. diff.
meanGINE	0.014 ± 0.008	$31.0\% \pm 8.0\%$
GIN	0.008 ± 0.007	$70.8\% \pm 8.7\%$
GINE	0.072 ± 0.074	$24.1\% \pm 18.0\%$
SignNet	0.280 ± 0.182	$10.1\% \pm 9.0\%$
DropGINE	0.020 ± 0.013	$39.4\% \pm 4.2\%$
ESAN	0.059 ± 0.039	$36.7\% \pm 9.1\%$
PPGN	0.020 ± 0.007	$5.9\% \pm 3.1\%$

AttrPGD - MolHIV

Model Type	Bruteforce - AttrPGD lbl. diff.	% of matched lbl. diff.
meanGINE	0.078 ± 0.035	$45.7\% \pm 1.7\%$
GINE	0.077 ± 0.035	$41.1\% \pm 12.2\%$
SignNet	0.213 ± 0.060	$14.3\% \pm 5.4\%$
DropGINE	0.089 ± 0.013	$24.4\% \pm 4.7\%$
ESAN	0.207 ± 0.119	$8.0\% \pm 4.2\%$
PPGN	0.063 ± 0.024	$4.2\% \pm 0.7\%$

Table 5.8: Effectiveness of AdjPGD and AttrPGD on budget 1

5.6 Impact of Adversarial Training

In this section, we present data for the random perturbation experiments, as well as brute force and PGD attacks on models that were trained on the MUTAG dataset, both using a typical ERM training procedure as well as using adversarial training.

We apply adversarial training as outlined in Section 4.6, effectively by training the model on the strongest adversarial examples in each epoch. To find such adversarial examples, we use the strongest attack AttrPGD with an absolute budget of 1. To improve the speed of the training, we reduce the number of PGD steps to just 3 and only sample 10 graphs from the obtained distributions. We fix the learning rate to the average of the values that were chosen during the hyperparameter optimization for the non-adversarially trained models. We do so because hyperparameter tuning is computationally expensive and adds much overhead, especially when the dataset is small. We only test the impact of adversarial training on the MUTAG dataset since adversarial training on the other datasets was too slow for our time budget.

The hyperparameters selection and model training were done with the same parameters as the other models.

Clean Model Performance

Model Type	acc. ↑	acc. (adv.) ↑	Δ ↑	lbl. diff. ↓	lbl. diff. (adv.) ↓	Δ ↓
Baseline	0.484 ± 0.146	-	-	0.462 ± 0.041	-	-
meanGINE	0.874 ± 0.047	0.747 ± 0.044	-0.127	0.172 ± 0.076	0.292 ± 0.032	0.120
GIN	0.947 ± 0.037	0.874 ± 0.060	-0.073	0.085 ± 0.024	0.186 ± 0.033	0.101
GINE	$\mathbf{0.989 \pm 0.024}$	0.853 ± 0.058	-0.136	$\mathbf{0.019 \pm 0.012}$	0.157 ± 0.048	0.138
SignNet	0.958 ± 0.058	0.726 ± 0.242	-0.232	0.153 ± 0.050	0.337 ± 0.125	0.184
DropGINE	0.947 ± 0.037	$\mathbf{0.916 \pm 0.029}$	$\mathbf{-0.031}$	0.054 ± 0.026	$\mathbf{0.102 \pm 0.018}$	**0.048**
ESAN	$\mathbf{0.979 \pm 0.029}$	$\mathbf{0.905 \pm 0.058}$	-0.074	$\mathbf{0.031 \pm 0.013}$	$\mathbf{0.118 \pm 0.043}$	0.087
PPGN	0.853 ± 0.044	0.779 ± 0.024	-0.074	0.176 ± 0.013	0.274 ± 0.026	0.098

Table 5.9: Clean MUTAG Model Metrics (↑: higher is better, ↓: lower is better)

The GINE and ESAN models obtain the best test scores when trained on clean data. As expected, all architectures perform worse on clean data when trained adversarially. The gap between standard and adversarial training is the smallest for DropGINE, which makes it the best-performing architecture on clean data under adversarial training, closely followed by ESAN. It is unexpected that PPGN, the only 3-WL-expressive architecture in the experiment, doesn't have the smallest gap. However, even the non-expressive architectures should be able to differentiate between original and attacked graphs for most perturbations of budget 1, especially if it is an attribute perturbation[12]. Another factor could be the dataset, which has a test set of only 19 graphs.

Adjacency Bruteforce

Model Type	acc. ↑	acc. (adv.) ↑	Δ acc. ↑	lbl. diff. ↓	lbl. diff. (adv.) ↓	Δ lbl. diff. ↓
meanGINE	0.084 ± 0.121	0.211 ± 0.064	0.127	0.874 ± 0.078	0.758 ± 0.036	−0.116
GIN	0.000 ± 0.000	0.189 ± 0.103	0.189	0.957 ± 0.013	0.760 ± 0.059	−0.197
GINE	0.189 ± 0.088	0.400 ± 0.060	0.211	0.796 ± 0.079	0.591 ± 0.055	−0.205
SignNet	0.021 ± 0.047	0.326 ± 0.256	0.305	0.922 ± 0.125	0.657 ± 0.153	−0.265
DropGINE	0.200 ± 0.164	0.495 ± 0.071	0.295	0.797 ± 0.150	0.522 ± 0.072	−0.275
ESAN	0.147 ± 0.114	0.516 ± 0.101	**0.369**	0.857 ± 0.093	0.508 ± 0.076	**−0.349**
PPGN	$\mathbf{0.589 \pm 0.094}$	$\mathbf{0.589 \pm 0.024}$	0.000	$\mathbf{0.488 \pm 0.107}$	$\mathbf{0.430 \pm 0.028}$	−0.058

Adjacency and Attribute Bruteforce

Model Type	acc. ↑	acc. (adv.) ↑	Δ acc.	lbl. diff. ↓	lbl. diff. (adv.) ↓	Δ lbl. diff.
meanGINE	0.011 ± 0.024	0.137 ± 0.029	0.126	0.975 ± 0.016	0.820 ± 0.029	0.155
GINE	0.158 ± 0.118	0.379 ± 0.044	0.221	0.825 ± 0.094	0.603 ± 0.049	0.222
SignNet	0.021 ± 0.047	0.305 ± 0.267	0.284	0.922 ± 0.125	0.676 ± 0.161	0.246
DropGINE	0.137 ± 0.142	0.432 ± 0.058	0.295	0.857 ± 0.119	0.565 ± 0.049	0.292
ESAN	0.105 ± 0.064	0.495 ± 0.109	**0.390**	0.894 ± 0.057	0.532 ± 0.068	**0.362**
PPGN	$\mathbf{0.558 \pm 0.109}$	$\mathbf{0.589 \pm 0.024}$	0.031	$\mathbf{0.515 \pm 0.095}$	$\mathbf{0.436 \pm 0.024}$	0.079

Table 5.10: MUTAG Bruteforce Results (↑: higher is better, ↓: lower is better)

Brute Force Attacks

All adversarially trained models outperform their standard-training counterparts under both types of brute force attacks, however, there is a stronger difference between less and more expressive architectures. GIN and meanGINE are the weakest models. The more-than 1-WL-expressive models beat the GIN(E)-based model, with SignNet being the only exception. Interestingly, PPGN performs best under all scenarios and even the unprotected models models outperform all adversarially trained models, with little difference between standard- and adversarially-trained PPGN models. However, the other expressive architectures achieve stronger accuracy and label difference improvements with adversarial training than the less expressive GIN(E)-based models.

Random Perturbations

The results in table 5.11 show that adversarial training mostly boosts CRR scores under random adjacency perturbations with a relative budget. PPGN achieves the highest CRR scores for both training types under random adjacency perturbations.

Even though the adversarial training uses AttrPGD to find perturbed graphs, the adversarially trained models don't seem to have significantly lower CRR scores under random attribute changes. Furthermore, the attribute perturbation results don't suggest a most robust architecture, however the adversarially trained GIN(E)-based models seem least robust.

[12]In graph-level MPNNs, attributes are combined in a series of MPNN layers, but ultimately aggregated into a graph-representing vector. This vector is processed using MLPs, which are universal approximators.

Adjacency Changes CRR - absolute budget

Model Type	acc.	acc. (adv.)	Δ acc.	lbl. diff.	lbl. diff. (adv.)	Δ lbl. diff.
meanGINE	**9.818 ± 6.097**	3.153 ± 0.478	−6.665	0.057 ± 0.041	0.016 ± 0.003	−0.041
GIN	3.135 ± 0.699	2.906 ± 0.319	−0.229	0.016 ± 0.002	0.018 ± 0.001	0.002
GINE	3.597 ± 0.309	5.512 ± 0.555	1.915	0.019 ± 0.001	0.032 ± 0.002	0.013
SignNet	3.001 ± 0.757	4.211 ± 3.617	1.210	0.013 ± 0.006	0.018 ± 0.019	0.005
DropGINE	7.135 ± 3.008	6.845 ± 2.968	−0.290	0.045 ± 0.017	0.045 ± 0.017	0.000
ESAN	4.321 ± 0.147	8.293 ± 1.296	3.972	0.023 ± 0.001	0.055 ± 0.005	0.032
PPGN	**10.806 ± 2.390**	**19.964 ± 3.946**	**9.158**	0.052 ± 0.012	**0.113 ± 0.031**	**0.061**

Adjacency Changes CRR - relative budget

Model Type	acc.	acc. (adv.)	Δ acc.	lbl. diff.	lbl. diff. (adv.)	Δ lbl. diff.
meanGINE	16.182 ± 6.879	7.370 ± 0.672	−8.812	0.102 ± 0.054	0.035 ± 0.004	−0.067
GIN	5.546 ± 0.526	7.102 ± 1.234	1.556	0.032 ± 0.003	0.041 ± 0.006	0.009
GINE	6.497 ± 0.501	12.149 ± 0.833	5.652	0.039 ± 0.004	0.081 ± 0.002	0.042
SignNet	4.442 ± 1.870	9.643 ± 9.128	5.201	0.021 ± 0.011	0.046 ± 0.053	0.025
DropGINE	5.445 ± 1.302	11.616 ± 0.801	6.171	0.031 ± 0.006	0.078 ± 0.004	0.047
ESAN	8.109 ± 0.418	**21.181 ± 2.568**	**13.072**	0.053 ± 0.003	**0.145 ± 0.019**	**0.092**
PPGN	**23.936 ± 3.370**	**23.715 ± 3.279**	−0.221	**0.157 ± 0.035**	**0.158 ± 0.028**	0.001

Attribute Changes CRR - absolute budget

Model Type	acc.	acc. (adv.)	Δ acc.	lbl. diff.	lbl. diff. (adv.)	Δ lbl. diff.
meanGINE	10.655 ± 5.491	2.611 ± 0.346	−8.044	0.063 ± 0.035	0.015 ± 0.002	−0.048
GIN	2.388 ± 0.646	3.394 ± 0.500	1.006	0.018 ± 0.005	0.019 ± 0.002	0.001
GINE	24.683 ± 1.420	13.030 ± 2.217	−11.653	**0.189 ± 0.013**	0.072 ± 0.011	−0.117
SignNet	**33.240 ± 6.997**	8.945 ± 5.926	−24.295	**0.209 ± 0.046**	0.027 ± 0.023	−0.182
DropGINE	21.607 ± 3.706	**18.709 ± 2.338**	−2.898	0.160 ± 0.036	**0.128 ± 0.021**	−0.032
ESAN	21.769 ± 3.184	**18.383 ± 3.879**	−3.386	0.160 ± 0.030	**0.120 ± 0.028**	−0.040
PPGN	16.180 ± 2.602	**18.064 ± 2.949**	**1.884**	0.081 ± 0.028	0.089 ± 0.028	**0.008**

Attribute Changes CRR - relative budget

Model Type	acc.	acc. (adv.)	Δ acc.	lbl. diff.	lbl. diff. (adv.)	Δ lbl. diff.
meanGINE	17.948 ± 6.270	5.975 ± 0.967	−11.973	0.119 ± 0.058	0.031 ± 0.007	−0.088
GIN	5.119 ± 2.103	7.649 ± 1.142	**2.530**	0.037 ± 0.016	0.044 ± 0.007	**0.007**
GINE	36.762 ± 2.779	24.910 ± 2.599	−11.852	**0.306 ± 0.027**	0.178 ± 0.018	−0.128
SignNet	**42.405 ± 5.501**	15.430 ± 12.100	−26.975	0.273 ± 0.051	0.059 ± 0.067	−0.214
DropGINE	36.279 ± 2.885	26.826 ± 1.903	−9.453	**0.304 ± 0.028**	0.207 ± 0.022	−0.097
ESAN	32.673 ± 2.186	**31.643 ± 2.047**	−1.030	0.267 ± 0.019	**0.234 ± 0.022**	−0.033
PPGN	23.290 ± 1.215	22.128 ± 3.723	−1.162	0.157 ± 0.011	0.135 ± 0.033	−0.022

Table 5.11: CRR scores under random perturbation (higher is better)

PGD-based Attacks

The results can be found in Fig. 5.4 on the next page.

All of the adversarially trained models except SignNet and PPGN are more robust than their counterparts that originated from standard training, although the performance of the adversarially trained SignNet is slightly better for some budgets. On AdjPGD, the unprotected PPGN is the most robust. The adversarially trained PPGN loses performance at similar rates as the unprotected version but ultimately performs worse due to its lower initial performance. Interestingly, PPGN is rivaled by the adversarially trained ESAN when attacked with AttrPGD.

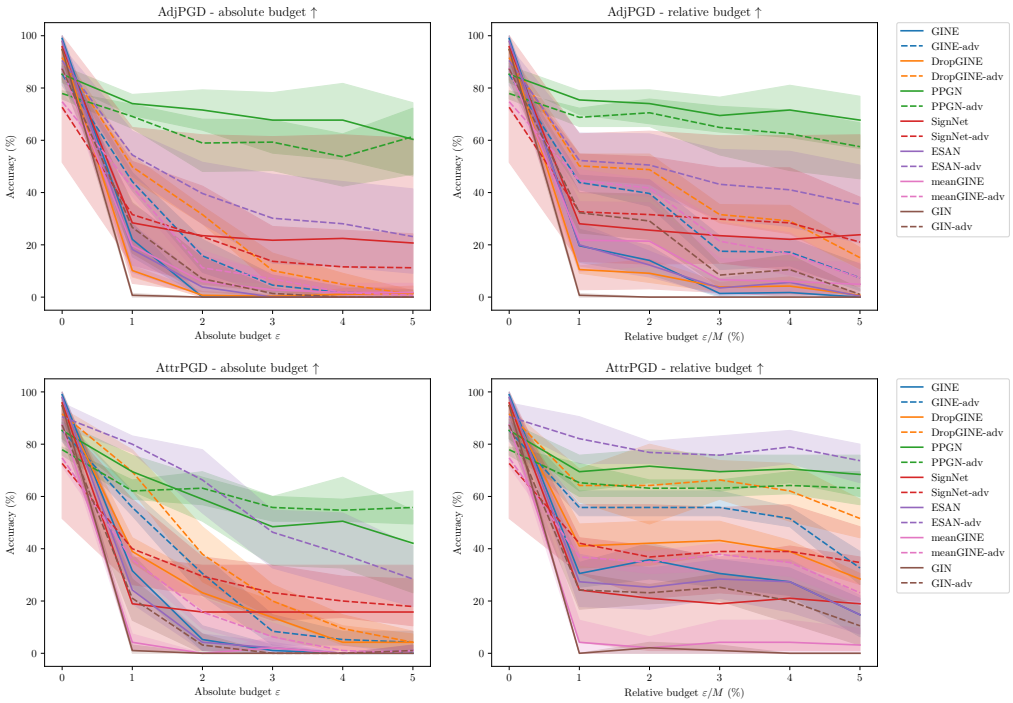


Figure 5.4: **MUTAG AdjPGD and AttrPGD**

Chapter 6

Conclusion and Future Work

We conclude by answering the questions posed in the introduction.

Are expressive GNNs more or less robust on graph-level tasks than less-expressive GNNs?

Concerning adversarial brute force and gradient-based attacks, the experimental data indicates that PPGN is the most robust architecture on a range of datasets compared to the other tested architectures.

PPGN is in many ways a special architecture. First of all, it achieves 3-WL-expressiveness and is thus the most expressive architecture in the hierarchy of WL-tests that we tested. Although it is not strictly more expressive than the other models[1], it should in principle be better equipped to differentiate between graphs than the other architectures. However, since expressivity is concerned with adjacency structure and not with attributes, we would expect architectures that are more robust due to their expressivity to perform similarly to non-expressive architectures under adversarial attribute perturbations. The fact that PPGN outperforms other architectures under attribute brute force and AttrPGD attacks indicates that its robustness may not be due to its expressivity[2]. The other expressive architectures, SignNet, DropGINE, and ESAN, only achieve good robustness irregularly (e.g. SignNet on IMDB under AdjPGD, ESAN on MolHIV under AttrPGD, DropGINE on ZINC12k under attribute brute force) and cannot differentiate themselves from the less-expressive GIN(E)-based models. This further implies that more expressive architectures are not necessarily more robust.

Another difference between PPGN and the other architectures is that it is natively differentiable with respect to the adjacency matrix. This allowed us to use the model as is in the attacks without the need for surrogate models. Since PGD-based attacks are least effective on PPGN (see tables 5.8 and A.5), they might benefit from special surrogate models that yield better gradients[3]. However, PPGN also appears robust under brute force attacks.

[1] In the sense that there may be graphs that cannot be differentiated by the 3-WL test and PPGN but by some other architecture. See [12] for an architecture that achieves strictly greater than 1-WL expressivity only on some pairs of graphs.

[2] Note that the adjacency and attribute attacks of PPGN prefer attribute changes, as shown in tables 5.5 and A.4.

[3] For example, [23] proposes a linearized surrogate model for graph convolutions to solve the attack optimization problem.

The reason for the robustness of PPGN may stem from its non-MPNN architecture. It operates on a dense representation of the input graph, which allows it to directly operate on adjacency information and combine it with attribute information into per-node-pair vectors. An adjacency change only modifies a few entries (the ones for the adjacency and the edge features) of each input node-pair vector in PPGN. In contrast, an adjacency change in MPNNs with sum aggregation alters every entry of the adjacent node embeddings after each layer, which might be less robust.

Regarding robustness under random perturbations, our data gives mixed results depending on the dataset with high uncertainties and thus doesn't allow establishing a hierarchy of robustness under random perturbations. Nonetheless, PPGN and SignNet appear unstable under high-budget random adjacency perturbations because they have extreme MAEs on the ZINC12k dataset.

In conclusion, our results do not indicate that the expressiveness of GNNs correlates with their robustness. However, they show that PPGN is a robust architecture across datasets for low-budget attacks.

How does the expressiveness of a model relate to robustness improvements due to adversarial training? Does the expressiveness of a model improve its performance on clean data when trained adversarially?

The unprotected PPGN achieves very high robustness on the MUTAG dataset, upon which the adversarially trained PPGN cannot improve. Excluding PPGN, the other expressive architectures (especially ESAN and DropGINE) achieve stronger accuracy and label difference improvements with adversarial training than the less expressive GIN(E)-based models, both under brute force and gradient-based attacks. This limited experiment thus indicates that more expressive models are indeed able to learn more robust decision boundaries.

As to the performance of adversarially trained models on clean data, there doesn't seem to be a significant improvement for the more expressive architectures, which could be a result of the small MUTAG test dataset.

Further Research

Our feature threat model allows for adjacency and feature changes but keeps the number of nodes fixed. As an extension, we could also consider adding or removing nodes as part of the threat model, which might be a reasonable attack vector in some domains. These changes could model the ability of an attacker to create or delete accounts in a social network. In the molecular domain, an attacker may add or remove nodes by adding or removing functional domains from the corresponding molecule.

Due to memory and time constraints, we weren't able to explore poisoning attacks on the expressive models. Especially the IMDB datasets, which classify movie genres based on actor collaborations, can be imagined in a setting where models are continuously trained in an online fashion, where an attacker is potentially able to poison the training data.

We also did not consider threat models that are specialized in the application domain. In the molecular domain, non-molecular graphs are easily detected and should thus not be part of a realistic threat model. Also, the notion of budget could be adapted to how similar molecules are, e.g. when analyzed by mass spectrometry, such that perturbations with a higher budget are more easily noticed. Besides the required domain knowledge, the challenge with specialized threat models is to find a formulation that allows for efficient attacks.

Bibliography

[1] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake news detection on social media using geometric deep learning," 2019.

[2] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, 2019, pp. 417–426.

[3] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[4] J. Xiong, Z. Xiong, K. Chen, H. Jiang, and M. Zheng, "Graph neural networks for automated de novo drug design," *Drug Discovery Today*, vol. 26, no. 6, pp. 1382–1393, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359644621000787

[5] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu, "Vision gnn: An image is worth graph of nodes," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8291–8303, 2022.

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

[7] P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer, "Dropgnn: Random dropouts increase the expressiveness of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 997–22 009, 2021.

[8] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bb04af0f7ecaee4aae62035497da1387-Paper.pdf

[9] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, "Equivariant subgraph aggregation networks," in *International Conference on Learning Representations*, 2022.

[10] D. Lim, J. Robinson, L. Zhao, T. Smidt, S. Sra, H. Maron, and S. Jegelka, "Sign and basis invariant networks for spectral graph representation learning," 2022.

[11] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[12] K. Martinkus, P. A. Papp, B. Schesch, and R. Wattenhofer, "Agent-based graph neural networks," in *The Eleventh International Conference on Learning Representations*, 2023.

[13] F. Mujkanovic, S. Geisler, S. Günnemann, and A. Bojchevski, "Are defenses for graph neural networks robust?" *Advances in Neural Information Processing Systems*, vol. 35, pp. 8954–8968, 2022.

[14] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[15] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJzIBfZAb

[16] J. Chen, X. Lin, H. Xiong, Y. Wu, H. Zheng, and Q. Xuan, "Smoothing adversarial training for gnn," *IEEE Transactions on Computational Social Systems*, vol. 8, no. 3, pp. 618–629, 2021.

[17] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[18] T. Sterling and J. J. Irwin, "Zinc 15 – ligand discovery for everyone," *Journal of Chemical Information and Modeling*, vol. 55, no. 11, pp. 2324–2337, 2015, pMID: 26479676. [Online]. Available: https://doi.org/10.1021/acs.jcim.5b00559

[19] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS Central Science*, vol. 4, no. 2, pp. 268–276, 2018, pMID: 29532027. [Online]. Available: https://doi.org/10.1021/acscentsci.7b00572

[20] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1365–1374. [Online]. Available: https://doi.org/10.1145/2783258.2783417

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

[22] K. Xu, H. Chen, S. Liu, P.-Y. Chen, T. W. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," in *International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence, 2019.

[23] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2847–2856. [Online]. Available: https://doi.org/10.1145/3219819.3220078

[24] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[25] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *International Conference on Learning Representations (ICLR)*, 2020.

[26] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," 1968.

[27] L. Babai, "Graph isomorphism in quasipolynomial time," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016, pp. 684–697.

[28] J.-Y. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identification," *Combinatorica*, vol. 12, no. 4, pp. 389–410, Dec 1992. [Online]. Available: https://doi.org/10.1007/BF01305232

[29] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec 2007. [Online]. Available: https://doi.org/10.1007/s11222-007-9033-z

[30] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bylnx209YX

[31] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. Ieee, 2017, pp. 39–57.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

Appendix A

Further Experimental Details

Model Type	MolHIV	ZINC12k	IMDB-BINARY	IMDB-MULTI	MUTAG
Baseline	328'501	282'001	280'501	281'103	275'701
meanGINE	966'906	920'406	918'906	922'518	914'106
GIN	966'906	920'406	-	-	914'106
GINE	966'906	920'406	918'906	922'518	914'106
SignNet	258'456	498'301	493'829	485'270	492'309
DropGINE	513'026	920'406	918'906	922'518	914'106
ESAN	105'829	96'201	95'885	96'143	94'899
PPGN	194'625	3'981'001	3'979'501	3'980'103	3'974'701

Table A.1: Parameter Count of Models

Node Feature	**Values**
Atom Number	1, ..., 118, misc
Chirality	unspecified, tetrahedral (CW / CCW), other, misc
Node Degree	0, ..., 10, misc
Formal Charge	-5, ..., 5, misc
# of Hs	0, ..., 8, misc
# of radical electrons	0, ..., 4, misc
Hybridization	SP, SP2, SP3, SP3D, SP3D2, misc
is aromatic	true, false
is in ring	true, false
Edge Feature	**Values**
Bond Type	single, double, triple, aromatic, misc
Stereo	stereo (none / z / e / cis / trans / any)
is conjugated	true, false

Table A.2: MolHIV Node and Edge Features

Model Type	**MolHIV** - label difference		**ZINC12k** - normalized MAE	
	abs. budget	rel. budget	abs. budget	rel. budget
meanGINE	0.784 ± 0.439	0.872 ± 0.453	0.878 ± 0.053	1.015 ± 0.105
GIN	0.154 ± 0.136	0.178 ± 0.145	0.709 ± 0.050	0.522 ± 0.052
GINE	0.026 ± 0.038	0.033 ± 0.046	1.031 ± 0.038	1.032 ± 0.102
SignNet	0.032 ± 0.013	0.701 ± 0.955	0.897 ± 0.118	0.631 ± 0.188
DropGINE	0.192 ± 0.044	0.482 ± 0.206	0.968 ± 0.066	0.894 ± 0.121
ESAN	0.051 ± 0.072	0.084 ± 0.116	1.178 ± 0.069	1.208 ± 0.124
PPGN	1.069 ± 0.700	1.163 ± 0.686	1.156 ± 0.033	1.028 ± 0.046

Model Type	**IMDB-BINARY**			
	absolute budget		**relative budget**	
	accuracy	lbl. diff.	accuracy	lbl. diff.
meanGINE	0.179 ± 0.128	0.072 ± 0.099	0.205 ± 0.167	0.100 ± 0.125
GINE	0.317 ± 0.174	0.179 ± 0.182	0.355 ± 0.119	0.213 ± 0.192
SignNet	5.309 ± 3.056	0.005 ± 0.011	5.497 ± 2.925	0.001 ± 0.003
DropGINE	1.154 ± 1.466	0.367 ± 0.080	1.151 ± 1.505	0.422 ± 0.206
ESAN	0.464 ± 0.243	0.305 ± 0.161	0.562 ± 0.372	0.375 ± 0.218
PPGN	2.572 ± 0.947	1.771 ± 0.964	3.096 ± 0.878	2.164 ± 0.909

Table A.3: CRR scores for random edge rewirings

Perturbation	**Target**	meanGINE	GIN	GINE	SignNet	DropGINE	ESAN	PPGN
added edge	increase	3.5%	2.7%	3.5%	2.7%	2.9%	1.2%	3.7%
	decrease	69.2%	66.9%	83.2%	82.6%	82.3%	86.0%	91.2%
dropped edge	increase	4.1%	1.6%	2.7%	6.1%	3.8%	4.7%	3.7%
	decrease	23.3%	28.9%	10.6%	8.5%	10.9%	8.0%	1.3%

Perturbation	**Target**	meanGINE	GINE	SignNet	DropGINE	ESAN	PPGN
adjacency	increase	0.2%	1.3%	3.6%	1.6%	1.5%	2.8%
	decrease	5.4%	19.2%	53.4%	21.3%	39.5%	39.2%
node attr.	increase	5.7%	4.3%	8.0%	4.1%	3.6%	6.2%
	decrease	88.2%	75.1%	34.1%	72.7%	55.0%	51.6%
edge attr.	increase	0.0%	0.0%	0.1%	0.1%	0.0%	0.0%
	decrease	0.5%	0.1%	0.8%	0.2%	0.4%	0.2%

Table A.4: Distribution of perturbation types amongst bruteforce attack on ZINC12k (original label)

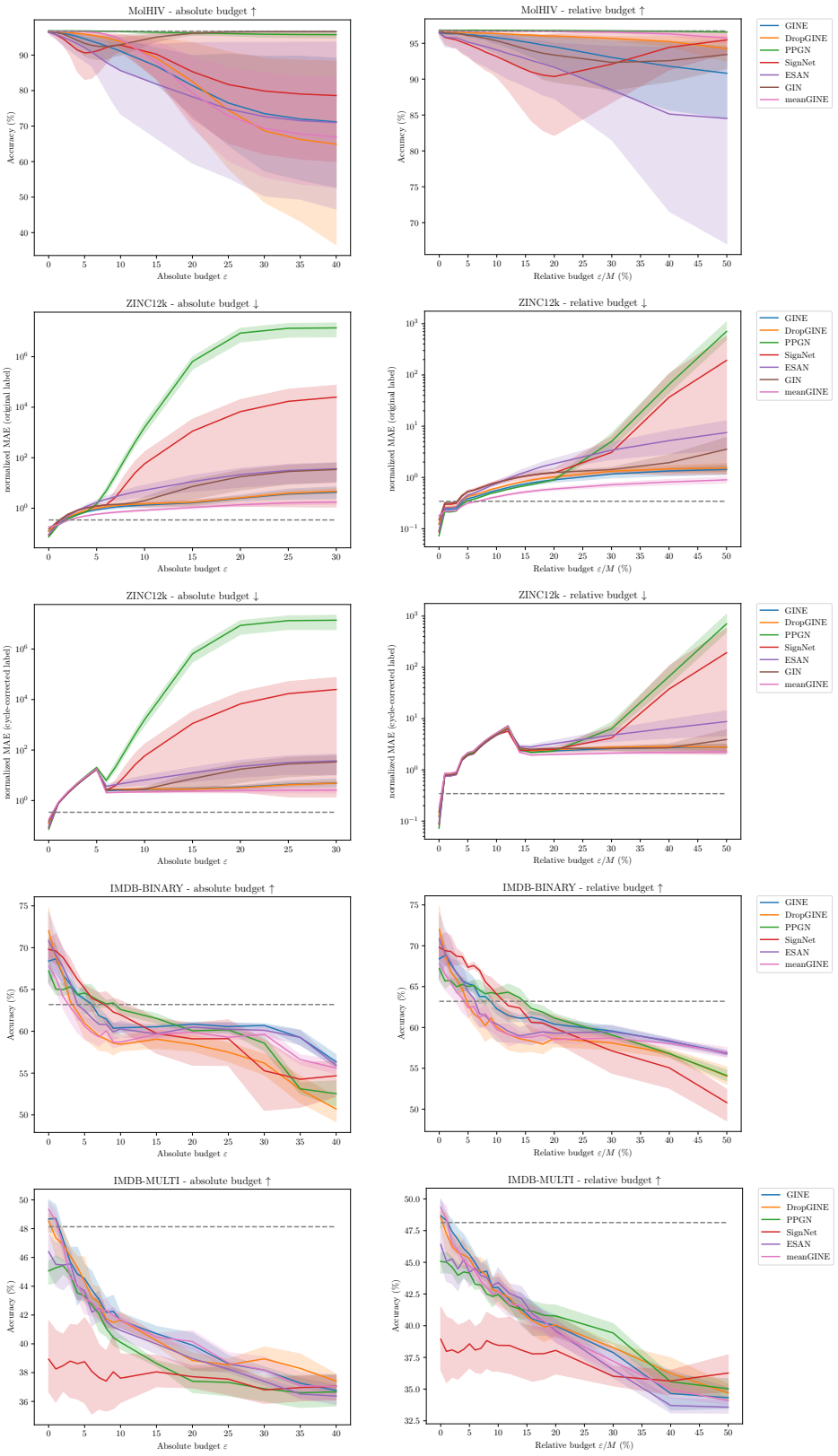


Figure A.1: **Random Adjacency Perturbations - Accuracy** Each point is the average over 5 models and 10 samples per graph (5 with added edges and 5 with removed edges). The standard deviation is over the 5 models.

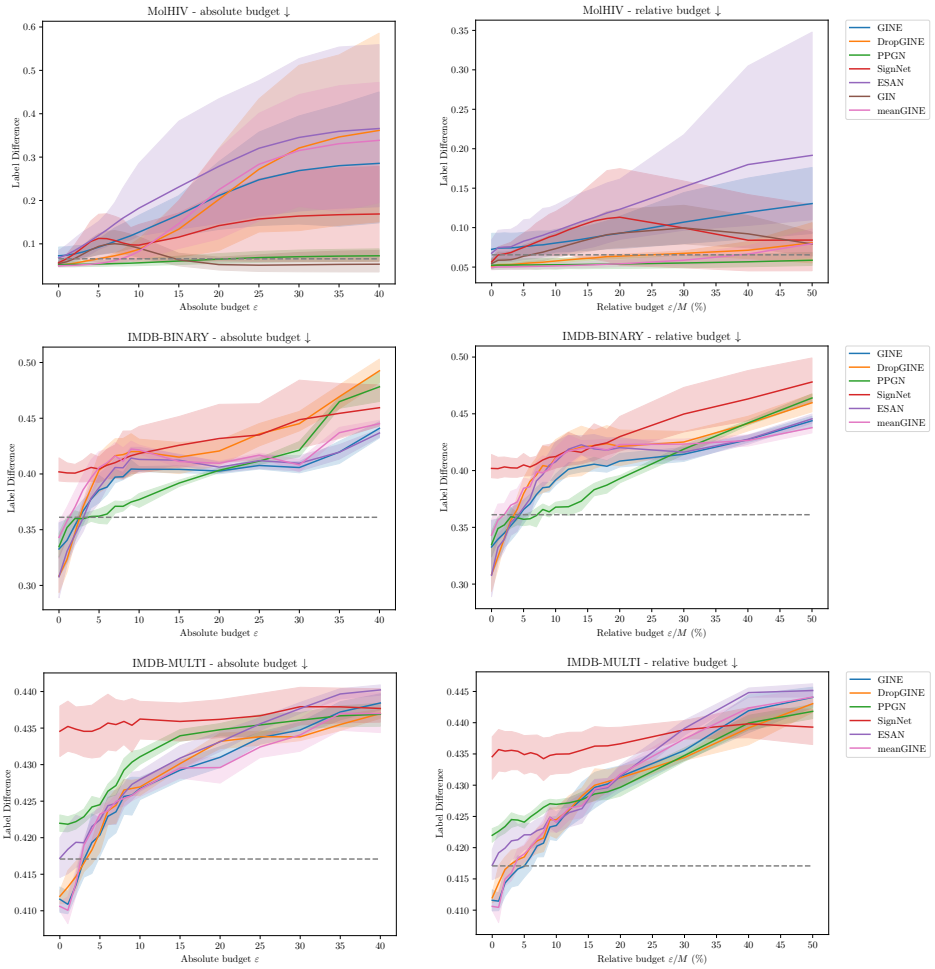


Figure A.2: **Random Adjacency Perturbations - Label Difference** Each point is the average over 5 models and 10 samples per graph (5 with added edges and 5 with removed edges). The standard deviation is over the 5 models.

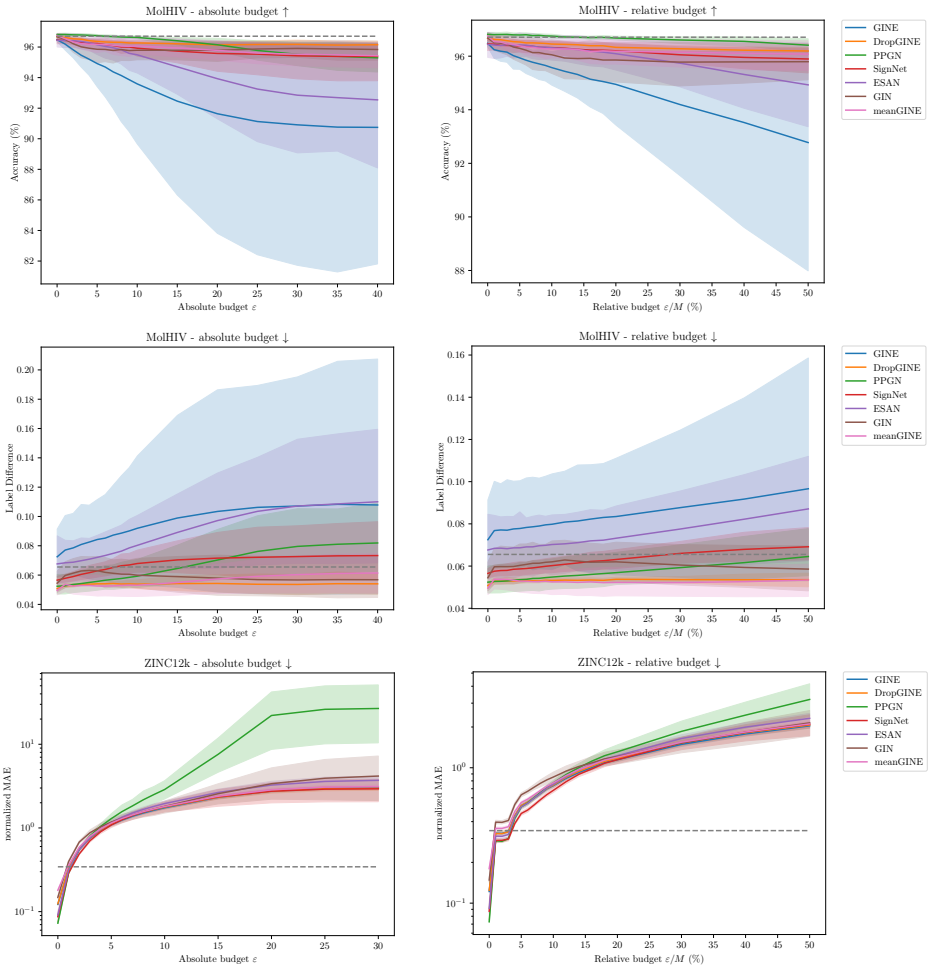


Figure A.3: **Random Attribute Perturbations** Each point is the average over 5 models and 10 samples per graph (5 with changed node features and 5 with changed edge features). The standard deviation is over the 5 models.

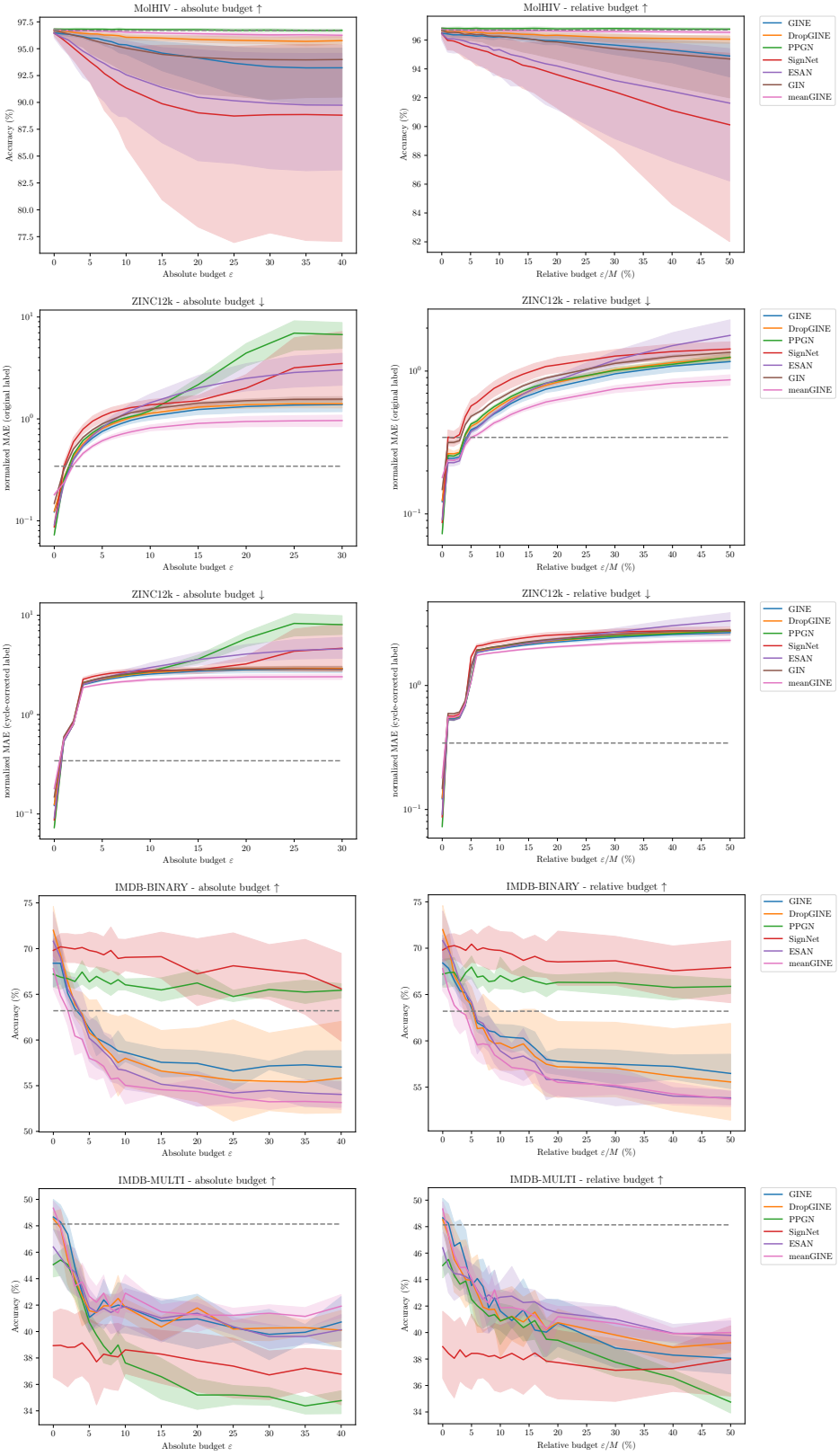


Figure A.4: **Random Edge Rewirings** Each point is the average over 5 models and 5 samples per graph. The standard deviation is over the 5 models.

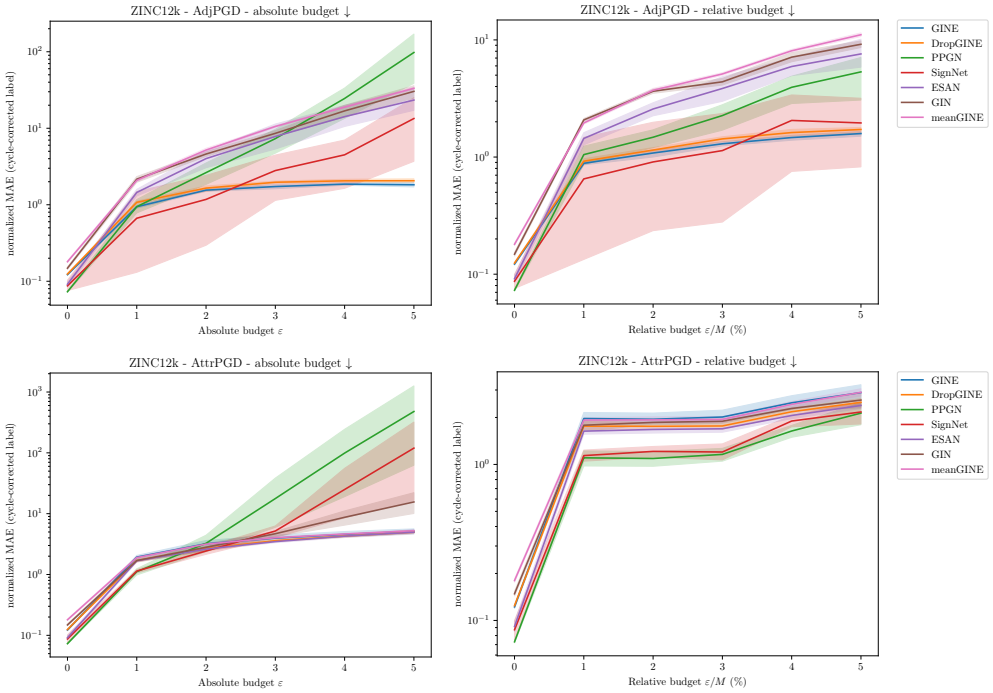


Figure A.5: **AdjPGD and AttrPGD on ZINC12k with cycle-corrected labels** (lower is better)

AdjPGD - ZINC12k (without discarding top 5%)

Model Type	AdjPGD / Bruteforce MAE	% of matched MAE
meanGINE	0.801 ± 0.049	$32.7\% \pm 7.4\%$
GIN	0.682 ± 0.082	$43.1\% \pm 9.4\%$
GINE	0.415 ± 0.034	$6.9\% \pm 1.5\%$
SignNet	0.318 ± 0.340	$19.0\% \pm 26.9\%$
DropGINE	0.502 ± 0.060	$10.6\% \pm 2.4\%$
ESAN	0.773 ± 0.066	$34.1\% \pm 6.1\%$
PPGN	0.160 ± 0.099	$3.0\% \pm 2.7\%$

AttrPGD - ZINC12k (without discarding top 5%)

Model Type	AttrPGD / Bruteforce MAE	% of matched MAE
meanGINE	0.783 ± 0.124	$25.6\% \pm 5.2\%$
GINE	0.783 ± 0.015	$38.7\% \pm 11.5\%$
SignNet	0.689 ± 0.130	$20.8\% \pm 17.4\%$
DropGINE	0.752 ± 0.019	$19.6\% \pm 2.8\%$
ESAN	0.719 ± 0.100	$12.8\% \pm 3.5\%$
PPGN	0.333 ± 0.188	$4.2\% \pm 3.2\%$

Table A.5: Effectiveness of AdjPGD and AttrPGD on budget 1 - ZINC12k

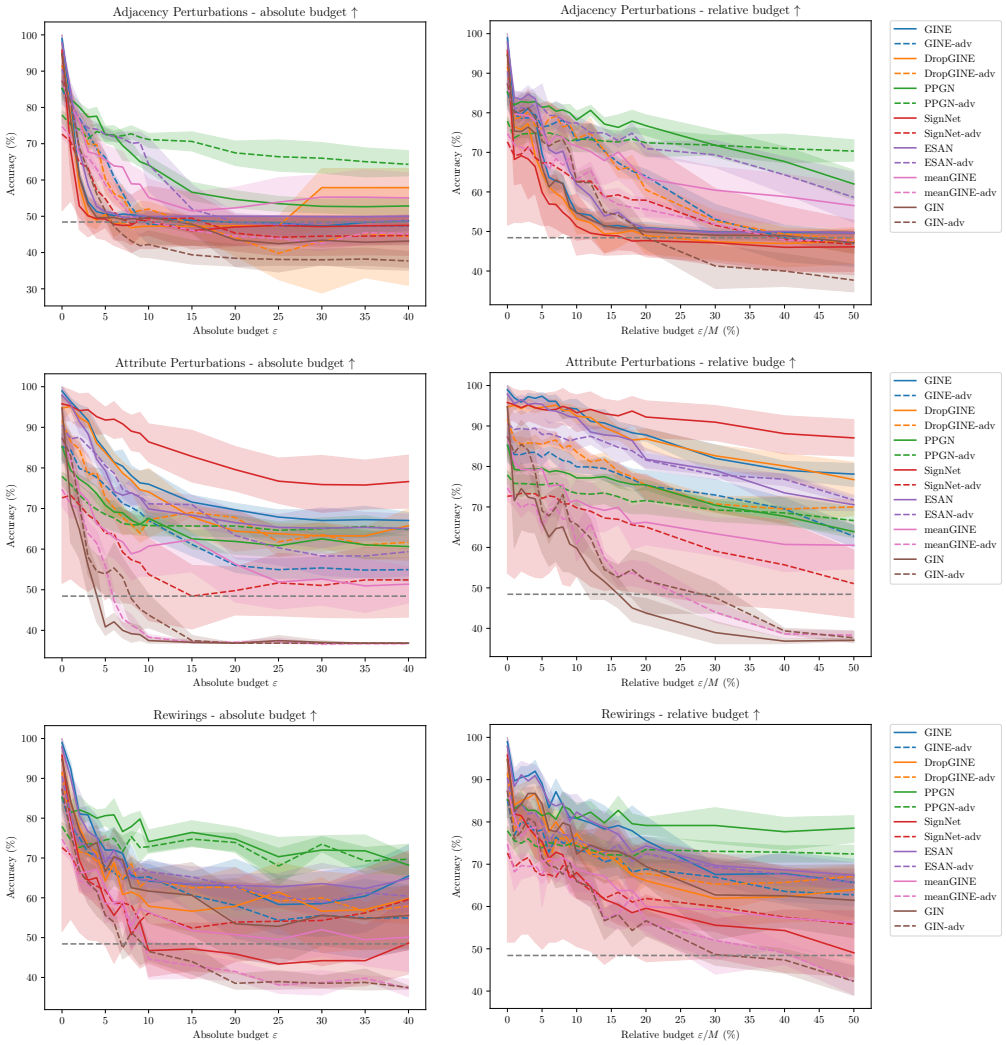


Figure A.6: **MUTAG random perturbations - Accuracy** (higher is better)

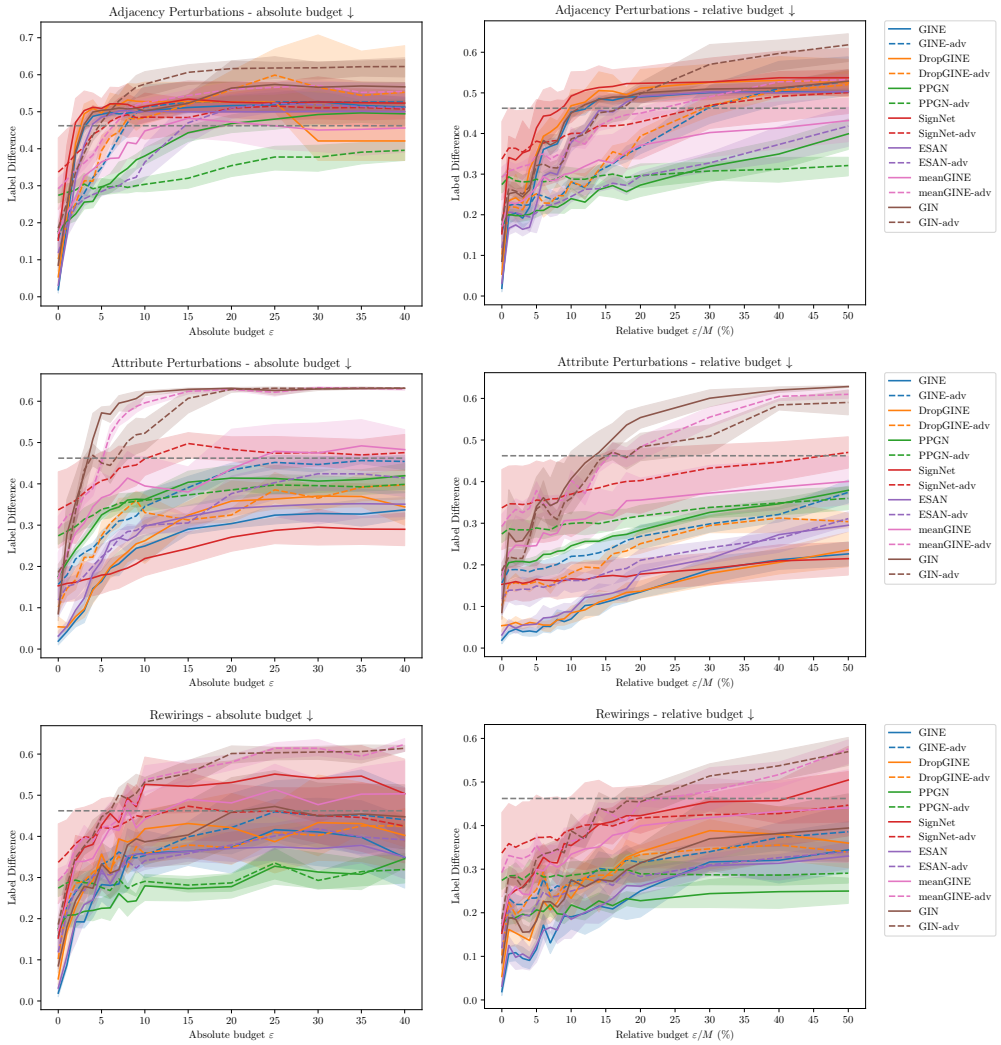


Figure A.7: **MUTAG random perturbations - Label Difference** (lower is better)

Rewiring - absolute budget

Model Type	acc.	acc. (adv.)	Δ acc.	lbl. diff.	lbl. diff. (adv.)	Δ lbl. diff.
meanGINE	7.682 ± 4.829	3.189 ± 0.437	-4.493	0.041 ± 0.029	0.016 ± 0.002	-0.025
GIN	12.770 ± 7.136	3.389 ± 0.615	-9.381	0.083 ± 0.052	0.019 ± 0.001	-0.064
GINE	17.356 ± 7.011	12.868 ± 4.776	-4.488	0.126 ± 0.057	0.082 ± 0.033	-0.044
SignNet	6.357 ± 4.196	11.353 ± 9.446	4.996	0.036 ± 0.026	0.052 ± 0.050	0.016
DropGINE	14.039 ± 7.016	14.285 ± 3.224	0.246	0.096 ± 0.057	0.088 ± 0.032	-0.008
ESAN	18.003 ± 3.289	16.010 ± 2.604	-1.993	0.125 ± 0.033	0.101 ± 0.025	-0.024
PPGN	25.905 ± 3.352	23.759 ± 3.079	-2.146	0.176 ± 0.033	0.163 ± 0.030	-0.013

Rewiring - relative budget

Model Type	acc.	acc. (adv.)	Δ acc.	lbl. diff.	lbl. diff. (adv.)	Δ lbl. diff.
DropGINE	19.946 ± 5.904	20.867 ± 2.165	0.921	0.147 ± 0.054	0.150 ± 0.018	0.003
ESAN	25.792 ± 1.860	22.914 ± 1.985	-2.878	0.201 ± 0.021	0.171 ± 0.017	-0.030
GIN	20.405 ± 7.608	7.860 ± 1.560	-12.545	0.148 ± 0.062	0.042 ± 0.006	-0.106
GINE	25.296 ± 4.297	19.942 ± 2.394	-5.354	0.199 ± 0.044	0.137 ± 0.025	-0.062
PPGN	31.397 ± 3.403	24.821 ± 2.969	-6.576	0.230 ± 0.032	0.174 ± 0.029	-0.056
SignNet	11.938 ± 2.073	15.220 ± 12.637	3.282	0.063 ± 0.020	0.069 ± 0.077	0.006
meanGINE	15.100 ± 7.332	8.202 ± 1.567	-6.898	0.093 ± 0.057	0.039 ± 0.004	-0.054

Table A.6: CRR scores under random edge rewirings (higher is better)