Cooperative Graph Neural Networks

Ben Finkelshtein [1] **Xingyue Huang** [1] **Michael Bronstein** [1] **İsmail İlkan Ceylan** [1]

Abstract

Graph neural networks are popular architectures for graph machine learning, based on iterative computation of node representations of an input graph through a series of invariant transformations. A large class of graph neural networks follow a standard message-passing paradigm: at every layer, each node state is updated based on an aggregate of messages from its neighborhood. In this work, we propose a novel framework for training graph neural networks, where every node is viewed as a *player* that can choose to either 'listen', 'broadcast', 'listen and broadcast', or to 'isolate'. The standard message propagation scheme can then be viewed as a special case of this framework where every node 'listens and broadcasts' to all neighbors. Our approach offers a more flexible and dynamic message-passing paradigm, where each node can determine its own strategy based on their state, effectively exploring the graph topology while learning. We provide a theoretical analysis of the new message-passing scheme which is further supported by an extensive empirical analysis on synthetic and real-world data.

1. Introduction

Graph neural networks (GNNs) (Scarselli et al., 2009; Gori et al., 2005) are a class of architectures for learning on graph-structured data. Their success in various graph machine learning (ML) tasks (Shlomi et al., 2021; Duvenaud et al., 2015; Zitnik et al., 2018) has led to a surge of different architectures (Kipf & Welling, 2017; Xu et al., 2019; Veličković et al., 2018; Hamilton et al., 2017). The vast majority of GNNs can be implemented through *message-passing*, where the fundamental idea is to update each node's representation based on an aggregate of messages flowing from the node's neighbors (Gilmer et al., 2017).

[1]Department of Computer Science, University of Oxford. Correspondence to: name surname <{name.surname}@cs.ox.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

The message-passing paradigm has been very influential in graph ML, but it also comes with well-known limitations related to the information flow on a graph, pertaining to *long-range* dependencies (Dwivedi et al., 2022). In order to receive information from k-hop neighbors, a network needs at least k layers, which typically implies an exponential growth of a node's receptive field. The growing amount of information needs then to be compressed into fixed-sized node embeddings, possibly leading to information loss, referred to as *over-squashing* (Alon & Yahav, 2021). Another well-known limitation related the information flow is *over-smoothing* (Li et al., 2018): the node features can become increasingly similar as the number of layers increases.

Motivation. Our goal is to generalize the message-passing scheme by allowing each node to decide how to propagate information *from* or *to* its neighbors, thus enabling a more flexible flow of information. Consider the example depicted in Figure 1, where the top row shows the information flow relative to the node u across three layers, and the bottom row shows the information flow relative to the node v across three layers. Node u listens to every neighbor in the first layer, only to v in the second layer, and to nodes s and r in the last layer. On the other hand, node v listens to node w for the first two layers, and to node u in the last layer. To realize this scenario, each node should be able to decide whether or not to listen to a particular node at each layer: a *dynamic* and *asynchronous* message-passing scheme, which clearly falls outside of standard message-passing.

Approach. To achieve this goal, we regard each node as a *player* that can take the following actions in each layer:

- STANDARD (S): Broadcast to neighbors that listen *and* listen to neighbors that broadcast.
- LISTEN (L): Listen to neighbors that broadcast.
- BROADCAST (B): Broadcast to neighbors that listen.
- ISOLATE (I): Neither listen nor broadcast.

When all nodes perform the action STANDARD, we recover the standard message-passing. Conversely, having all the nodes ISOLATE corresponds to removing all the edges from the graph implying node-wise predictions. The interplay between these actions and the ability to change them *dynamically* makes the overall approach richer and allows to decouple the input graph from the computational one and incorporate directionality into message-passing: a node can

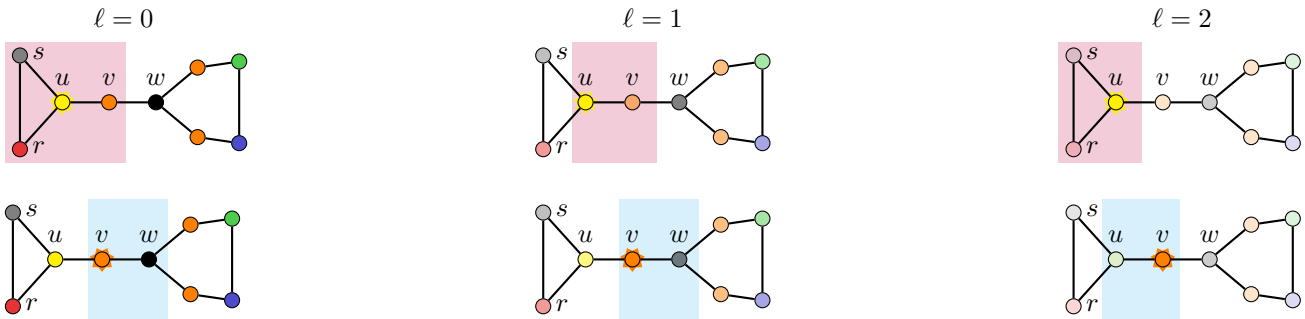


Figure 1: Example information flow for nodes u, v. **Top**: information flow relative to u across three layers. Node u listens to every neighbor in the first layer, but only to v in the second layer, and only to s and r in the last layer. **Bottom**: information flow relative to v across three layers. The node v listens only to w in the first two layers, and only to u in the last layer.

only listen to those neighbors that are currently broadcasting, and vice versa. We can emulate the example from Figure 1 by making u choose the actions $\langle \mathrm{L}, \mathrm{L}, \mathrm{S} \rangle$, v and w the actions $\langle \mathrm{S}, \mathrm{S}, \mathrm{L} \rangle$, and s and r the actions $\langle \mathrm{S}, \mathrm{I}, \mathrm{S} \rangle$.

Contributions. We develop a new class of architectures, dubbed *cooperative graph neural networks* (CO-GNNs), where every node in the graph is viewed as a player that can perform one of the aforementioned actions. CO-GNNs comprise two jointly trained "cooperating" message-passing neural networks: an *environment network* η (for solving the given task), and an *action network* π (for choosing the best actions). Our contributions can be summarized as follows:

- We propose a novel message-passing mechanism, which leads to CO-GNN architectures that effectively explore the graph topology while learning (Section 4).
- We provide a detailed discussion on the properties of CO-GNNs (Section 5.1) and show that they are more expressive than 1-dimensional Weisfeiler-Leman algorithm (1-WL) (Section 5.2), and better suited for long-range tasks due to their adaptive nature (Section 5.3).
- Empirically, we focus on CO-GNNs with *basic* action and environment networks to carefully assess the virtue of the new message-passing paradigm. We first validate the strength of our approach on a synthetic task (Section 6.1). Then, we conduct experiments on real-world datasets, and observe that CO-GNNs always improve compared to their baseline models, and yield multiple state-of-the-art results (Section 6.2 and Appendix C.3).
- We compare the trend of the actions on homophilic and heterophilic graphs (Section 7.1); visualize the actions on a heterophilic graph (Section 7.2); and ablate on the choices of action and environment networks (Section 7.3). We complement these with experiments related to expressive power (Appendix C.1), long-range tasks (Appendix C.2), and over-smoothing (Appendix C.5).

Additional details can be found in the appendix of this paper.

2. Background

Graph Neural Networks. We consider simple, undirected attributed graphs $G = (V, E, \boldsymbol{X})$, where $\boldsymbol{X} \in \mathbb{R}^{|V| \times d}$ is a matrix of (input) node features, and $\boldsymbol{x}_v \in \mathbb{R}^d$ denotes the feature of a node $v \in V$. We focus on *message-passing neural networks (MPNNs)* (Gilmer et al., 2017) that encapsulate the vast majority of GNNs. An MPNN updates the initial node representations $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$ of each node v for $0 \leq \ell \leq L-1$ iterations based on its own state and the state of its neighbors $\mathcal{N}_v$ as:

$$\boldsymbol{h}_v^{(\ell+1)} = \phi^{(\ell)} \left(\boldsymbol{h}_v^{(\ell)}, \psi^{(\ell)} \left(\boldsymbol{h}_v^{(\ell)}, \{\!\!\{ \boldsymbol{h}_u^{(\ell)} \mid u \in \mathcal{N}_v \}\!\!\} \right) \right),$$

where $\{\!\!\{ \cdot \}\!\!\}$ denotes a multiset and $\phi^{(\ell)}$ and $\psi^{(\ell)}$ are differentiable *update* and *aggregation* functions, respectively. We denote by $d^{(\ell)}$ the dimension of the node embeddings at iteration (layer) ℓ. The final representations $\boldsymbol{h}_v^{(L)}$ of each node v can be used for predicting node-level properties or they can be pooled to form a graph embedding vector $\boldsymbol{z}_G^{(L)}$, which can be used for predicting graph-level properties. The pooling often takes the form of simple averaging, summation, or element-wise maximum. Of particular interest to us are the basic MPNNs:

$$\boldsymbol{h}_v^{(\ell+1)} = \sigma \left(\boldsymbol{W}_s^{(\ell)} \boldsymbol{h}_v^{(\ell)} + \boldsymbol{W}_n^{(\ell)} \psi \left(\{\!\!\{ \boldsymbol{h}_u^{(\ell)} \mid u \in \mathcal{N}_v \}\!\!\} \right) \right),$$

where $\boldsymbol{W}_s^{(\ell)}$ and $\boldsymbol{W}_n^{(\ell)}$ are $d^{(\ell)} \times d^{(\ell+1)}$ learnable parameter matrices acting on the node's self-representation and on the aggregated representation of its neighbors, respectively, σ is a non-linearity, and ψ is either *mean* or *sum* aggregation function. We refer to the architecture with mean aggregation as MEANGNNs and to the architecture with sum aggregation as SUMGNNs (Hamilton, 2020). We also consider prominent models such as GCN (Kipf & Welling, 2017), GIN (Xu et al., 2019) and GAT (Veličković et al., 2018).

Straight-through Gumbel-softmax Estimator. In our approach, we rely on an action network for predicting categorical actions for the nodes in the graph, which is not

differentiable and poses a challenge for gradient-based optimization. One prominent approach to address this is given by the Gumbel-softmax estimator (Jang et al., 2017; Maddison et al., 2017) which effectively provides a differentiable, continuous approximation of discrete action sampling. Consider a finite set Ω of actions. We are interested in learning a categorical distribution over Ω, which can be represented in terms of a probability vector $\boldsymbol{p} \in \mathbb{R}^{|\Omega|}$ whose elements store the probabilities of different actions. Let us denote by $\boldsymbol{p}(a)$ the probability of an action $a \in \Omega$. Gumbel-softmax is a special reparametrization trick that estimates the categorical distribution $\boldsymbol{p} \in \mathbb{R}^{|\Omega|}$ with the help of a Gumbel-distributed vector $\boldsymbol{g} \in \mathbb{R}^{|\Omega|}$, which stores an i.i.d. sample $\boldsymbol{g}(a) \sim \text{GUMBEL}(0,1)$ for each action a. Given a categorical distribution $\boldsymbol{p}$ and a temperature parameter τ, Gumbel-softmax (GS) scores can be computed as follows:

$$\text{GS}\left(\boldsymbol{p};\tau\right) = \frac{\exp\left(\left(\log(\boldsymbol{p}) + \boldsymbol{g}\right)/\tau\right)}{\sum_{a\in\Omega}\exp\left(\left(\log(\boldsymbol{p}(a)) + \boldsymbol{g}(a)\right)/\tau\right)}$$

As the softmax temperature τ decreases, the resulting vector tends to a *one-hot* vector. Straight-through GS estimator utilizes the GS estimator during the backward pass only (for a differentiable update), while during the forward pass, it employs an ordinary sampling.

3. Related Work

Most of GNNs operate by message-passing (Gilmer et al., 2017), including architectures such as GCNs (Kipf & Welling, 2017), GIN (Xu et al., 2019), GAT (Veličković et al., 2018), and GraphSAGE (Hamilton et al., 2017). Despite their success, MPNNs have some known limitations.

First, the expressive power of MPNNs is upper bounded by 1-WL (Xu et al., 2019; Morris et al., 2019). This motivated the study of more expressive architectures, based on higher-order structures (Morris et al., 2019; Maron et al., 2019; Keriven & Peyré, 2019), subgraph (Bevilacqua et al., 2022; Thiede et al., 2021) or homomorphism counting (Barceló et al., 2021; Jin et al., 2024), node features with unique identifiers (Loukas, 2020), or random features (Abboud et al., 2021; Sato et al., 2021).

Second, MPNNs perform poorly on long-range tasks due to their information propagation bottlenecks such as *over-squashing* (Alon & Yahav, 2021) and *over-smoothing* (Li et al., 2018). The former limitation motivated approaches based on rewiring the graph (Klicpera et al., 2019; Topping et al., 2022; Karhadkar et al., 2023) by connecting relevant nodes and shortening propagation distances, or designing new message-passing architectures that act on distant nodes directly, e.g., using shortest-path distances (Abboud et al., 2022; Ying et al., 2021). The over-smoothing problem has also motivated a body of work to avoid the collapse of node features (Zhao & Akoglu, 2019; Chen et al., 2020).

Finally, classical message passing updates the nodes in a fixed and synchronous manner, which does not allow the nodes to react to messages from their neighbors individually. This has been recently argued as yet another limitation of classical message passing from the perspective of algorithmic alignment (Faber & Wattenhofer, 2022).

Our approach presents new perspectives on these limitations via a dynamic and asynchronous information flow (see Section 5). This is related to the work of Lai et al. (2020), where the goal is to update each node using a different number of layers (over a fixed topology). This is also related to the work of (Dai et al., 2022), where the idea is to apply message passing on a learned topology but one that is the same at every layer. CO-GNNs diverge from these studies in terms of the objectives and the approach.

4. Cooperative Graph Neural Networks

CO-GNNs view each node in a graph as a *player* of a multi-player environment, where the state of each player is given in terms of the representation (or *state*) of its corresponding node. Every node is updated following a two-stage process. In the first stage, each node chooses an action from the set of actions given their current state and the states of their neighboring nodes. In the second stage, every node state gets updated based on their current state and the states of a *subset* of the neighboring nodes, as determined by the actions in the first stage. As a result, every node can determine how to propagate information from or to its neighbors.

A CO-GNN (π, η) architecture is given in terms of two cooperating GNNs: (i) an action network π for choosing the best actions, and (ii) an environment network η for updating the node representations. A CO-GNN layer updates the representations $\boldsymbol{h}_v^{(\ell)}$ of each node v as follows. First, an action network π predicts, for each node v, a probability distribution $\boldsymbol{p}_v^{(\ell)} \in \mathbb{R}^4$ over the actions $\{\text{S}, \text{L}, \text{B}, \text{I}\}$ that v can take, given its state and the state of its neighbors $\mathcal{N}_v$:

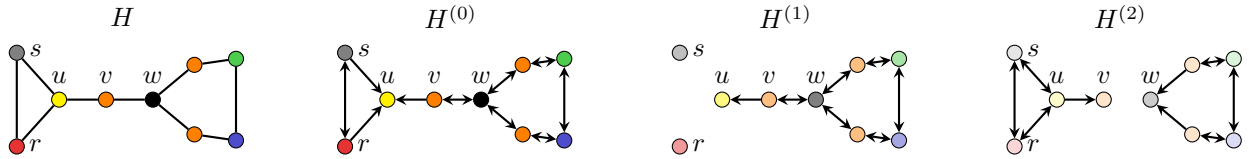
$$\boldsymbol{p}_v^{(\ell)} = \pi\left(\boldsymbol{h}_v^{(\ell)}, \{\!\!\{\boldsymbol{h}_u^{(\ell)} \mid u \in \mathcal{N}_v\}\!\!\}\right). \tag{1}$$

Then, for each node v, an action is sampled $a_v^{(\ell)} \sim \boldsymbol{p}_v^{(\ell)}$ using Straight-through GS, and an environment network η is utilized to update the state of each node in accordance with the sampled actions:

$$\boldsymbol{h}_v^{(\ell+1)} = \begin{cases} \eta^{(\ell)}\left(\boldsymbol{h}_v^{(\ell)}, \{\!\!\{\}\!\!\}\right), & a_v^{(\ell)} = \text{I} \vee \text{B} \\ \eta^{(\ell)}\left(\boldsymbol{h}_v^{(\ell)}, \mathcal{M}\right), & a_v^{(\ell)} = \text{L} \vee \text{S} \end{cases} \tag{2}$$

where $\mathcal{M} = \{\!\!\{\boldsymbol{h}_u^{(\ell)} \mid u \in \mathcal{N}_v, a_u^{(\ell)} = \text{S} \vee \text{B}\}\!\!\}$.

This is a single layer update, and by stacking $L \geq 1$ layers, we obtain the representations $\boldsymbol{h}_v^{(L)}$ for each node v.



(a) Input graph H. (b) Computation at $\ell = 0$. (c) Computation at $\ell = 1$. (d) Computation at $\ell = 2$.

Figure 2: The input graph H and its computation graphs $H^{(0)}$, $H^{(1)}$, $H^{(2)}$ that are a result of applying the actions: $\langle \mathrm{L}, \mathrm{L}, \mathrm{S} \rangle$ for the node u; $\langle \mathrm{S}, \mathrm{S}, \mathrm{L} \rangle$ for the nodes v and w; $\langle \mathrm{S}, \mathrm{I}, \mathrm{S} \rangle$ for the nodes s and r; $\langle \mathrm{S}, \mathrm{S}, \mathrm{S} \rangle$ for all other nodes.

In its full generality, a CO-GNN (π, η) architecture can operate on (un)directed graphs and use any GNN architecture in place of the action network π and the environment network η. To carefully assess the virtue of this new message-passing paradigm, we consider relatively simple architectures such as SUMGNNs, MEANGNNs, GCN, GIN and GAT, which are respectively denoted as $\sum$, μ, $*$, ϵ, and α. For example, we write CO-GNN(Σ, μ) to denote a CO-GNN architecture which uses SUMGNN as its action network and MEANGNN as its environment network.

Fundamentally, CO-GNNs update the node states in a fine-grained manner: if a node v chooses to ISOLATE or to BROADCAST then it gets updated only based on its previous state, which corresponds to a node-wise update function. On the other hand, if a node v chooses the action LISTEN or STANDARD then it gets updated based on its previous state as well as the state of its neighbors which perform the actions BROADCAST or STANDARD at this layer.

5. Model Properties

We analyze CO-GNNs, focusing on conceptual novelty, expressive power, and suitability to long-range tasks.

5.1. Conceptual Properties

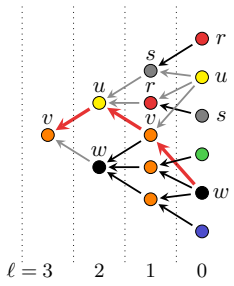
Task-specific: Standard message-passing updates nodes based on their neighbors, which is completely task-agnostic. By allowing each node to listen to the information from 'relevant' neighbors only, CO-GNNs can determine a computation graph which is best suited for the target task. For example, if the task requires information only from the neighbors with a certain degree then the action network can learn to listen only to these nodes (see Section 6.1).

Directed: The outcome of the actions that the nodes can take amounts to a special form of *'directed rewiring'* of the input graph: an edge can be *dropped* (e.g., if two neighbors listen without broadcasting); an edge can remain *undirected* (e.g., if both neighbors apply the standard action); or, an edge can *become directed* implying directional information flow (e.g., if one neighbor listens while its neighbor broadcasts). Taking this perspective, the proposed message-passing can be seen as operating on a potentially different directed graph induced by the choice of actions at every layer (illustrated in Figure 2). Formally, given a graph $G = (V, E)$, let us denote by $G^{(\ell)} = (V, E^{(\ell)})$ the directed computational graphs induced by the actions chosen at layer ℓ, where $E^{(\ell)}$ is the set of directed edges at layer ℓ. We can rewrite the update given in Equation (2) concisely as follows:

$$\boldsymbol{h}_v^{(\ell+1)} = \eta^{(\ell)} \left(\boldsymbol{h}_v^{(\ell)}, \{\!\!\{ \boldsymbol{h}_u^{(\ell)} \mid (u, v) \in E^{(\ell)} \}\!\!\} \right).$$

Consider the input graph H from Figure 2: u gets messages from v only in the first two layers, and v gets messages from u only in the last layer, illustrating a directional message-passing between these nodes. This abstraction allows for a direct implementation of CO-GNNs by simply considering the induced graph adjacency matrix at every layer.

Dynamic: In CO-GNNs, each node interacts with the 'relevant' neighbors and does so only as long as they remain relevant. CO-GNNs do not operate on a fixed computational graph, but rather on a learned computational graph, which is dynamic across layers. In our running example, the computational graph is a different one at every layer (depicted on the right hand side): This is advantageous for the information flow (see Section 5.3).



Feature and Structure Based: Standard message-passing is determined by the structure of the graph: two nodes with the same neighborhood get the same aggregated message. This is not necessarily the case in our setup, since the action network can learn different actions for two nodes with different node features, e.g., by choosing different actions for a *red* node and a *blue* node. This enables different messages for different nodes even if their neighborhoods are identical.

Asynchronous: Standard message-passing updates all nodes synchronously, which is not always optimal as argued by Faber & Wattenhofer (2022), especially when the task requires to treat the nodes non-uniformly. By design, CO-GNNs enable asynchronous updates across nodes.

Conditional Aggregation: The action network of Co-GNNs can be viewed as a look-ahead function that makes decisions after applying k layers. Specifically, at layer ℓ, an action network of depth k computes node representations on the original graph topology, which are $(k+\ell)$-layer representations. Based on these representations, the action network determines an action for each node, which induces a new graph topology for the environment network to operate on. In this sense, the aggregation of environment network at layer ℓ is determined by $(k+\ell)$-layer representations of the action network, which can be viewed as a "look-ahead" capability and the aggregation mechanism of the environment network is conditioned on this look-ahead capability.

Orthogonal to Attention: The (soft) attention mechanism on graphs allows for aggregating — based on learnable attention coefficients — a weighted mean of the features of neighboring nodes. While these architectures can weigh the contribution of different neighbors, they have certain limitations, e.g., weighted mean aggregation cannot count node degrees. Moreover, the conditional aggregation mechanism of Co-GNNs goes beyond the capabilities of attention-based architectures. The contribution of Co-GNNs is hence orthogonal to that of attention-based architectures, such as GATs, and these architectures can be used as base action/environment architectures in Co-GNNs. In Section 6.1, we empirically validate this via a task that GAT cannot solve, but Co-GNNs with a GAT environment network can.

Mitigates Over-smoothing: In principle, the action network of Co-GNNs can choose the action ISOLATE for a node if the features of the neighbours of this node are not informative. As a result, Co-GNNs can mitigate over-smoothing. We validate this empirically in Appendix C.5, but we also note that the optimisation becomes increasingly difficult once the number of layers gets too large.

Efficient: While being more sophisticated, our approach is efficient in terms of runtime, as we detail in Appendix D. Co-GNNs are also parameter-efficient: they share the same action network across layers and as a result a comparable number of parameters to their baseline models.

5.2. Expressive Power of Co-GNNs

The environment and action networks of Co-GNN architectures are parameterized by standard MPNNs. This raises an obvious question regarding the expressive power of Co-GNN architectures: *are* Co-GNN*s also bounded by 1-WL in terms of distingushing graphs?*

Proposition 5.1. *Let $G_1 = (V_1, E_1, \boldsymbol{X}_1)$ and $G_2 = (V_2, E_2, \boldsymbol{X}_2)$ be two non-isomorphic graphs. Then, for any threshold $0 < \delta < 1$, there exists a parametrization of a* Co-GNN *architecture using sufficiently many layers L, satisfying $\mathbb{P}(\boldsymbol{z}_{G_1}^{(L)} \neq \boldsymbol{z}_{G_2}^{(L)}) \geq 1 - \delta$.*

The explanation for this result is the following: Co-GNN architectures learn, at every layer, and for each node u, a probability distribution over the actions. These learned distributions are identical for two isomorphic nodes. However, the process relies on *sampling* actions from these distributions, and clearly, the samples from identical distributions can differ. This makes Co-GNN models *invariant in expectation,* and the variance introduced by the sampling process helps to discriminate nodes that are 1-WL indistinguishable. Thus, for two nodes indistinguishable by 1-WL, there is a non-trivial probability of sampling a different action for the respective nodes, which in turn makes their direct neighborhood differ. This yields unique node identifiers (Loukas (2020)) with high probability and allows us to distinguish any pair of graphs assuming an injective graph pooling function (Xu et al., 2019). This is analogous to GNNs with random node features (Abboud et al., 2021; Sato et al., 2021), which are more expressive than their classical counterparts. We validate the stated expressiveness gain in Appendix C.1.

It is important to note that Co-GNNs are *not* designed for expressiveness, and our result relies merely on variations in the sampling process, which is unstable and should be noted as a limitation. Clearly, Co-GNNs can also use more expressive architectures as base architectures.

5.3. Dynamic Message-passing for Long-range Tasks

Long-range tasks necessitate to propagate information between distant nodes: Co-GNNs are effective for such tasks since they can propagate only relevant task-specific information. Suppose that we are interested in transmitting information from a source node to a distant target node: Co-GNNs can efficiently filter irrelevant information by learning to focus on a path connecting these two nodes, hence maximizing the information flow to the target node. We can generalize this observation towards receiving information from multiple distant nodes and prove the following:

Proposition 5.2. *Let $G = (V, E, \boldsymbol{X})$ be a connected graph with node features. For some $k > 0$, for any target node $v \in V$, for any k source nodes $u_1, \ldots, u_k \in V$, and for any compact, differentiable function $f : \mathbb{R}^{d^{(0)}} \times \ldots \times \mathbb{R}^{d^{(0)}} \to \mathbb{R}^d$, there exists an L-layer* Co-GNN *computing final node representations such that for any $\epsilon, \delta > 0$ it holds that $\mathbb{P}(|\boldsymbol{h}_v^{(L)} - f(\boldsymbol{x}_{u_1}, \ldots \boldsymbol{x}_{u_k})| < \epsilon) \geq 1 - \delta$.*

This means that if a property of a node v is a function of k distant nodes then Co-GNNs can approximate this function. This follows from two findings: (i) the features of k nodes can be transmitted to the source node without loss of information and (ii) the final layer of a Co-GNN architecture, e.g., an MLP, can approximate any differentiable function over k node features (Hornik, 1991; Cybenko, 1989). We validate these findings empirically on long-range interactions datasets (Dwivedi et al., 2022) in Appendix C.2.

Figure 3: ROOTNEIGHBORS examples. **Left:** Example tree for ROOTNEIGHBORS. **Right:** Example of an optimal directed subgraph over the input tree, where the nodes with a degree of 6 (u and v) BROADCAST, while other nodes LISTEN.

6. Experimental Results

We evaluate CO-GNNs on a synthetic experiment, and on real-world node classification datasets (Platonov et al., 2023). We also report a synthetic expressiveness experiment, an experiment on long-range interactions datasets (Dwivedi et al., 2022), and graph classification datasets (Morris et al., 2020) in Appendix C. Our codebase is available at `https://github.com/benfinkelshtein/CoGNN`.

6.1. Synthetic Experiment on ROOTNEIGHBORS

Task. In this experiment, we compare CO-GNNs to MPNNs on a new dataset: ROOTNEIGHBORS. We consider the following regression task: *given a rooted tree, predict the average of the features of root-neighbors of degree* 6. This task requires to first identify the neighbors of the root node with degree 6 and then to return the average feature of these nodes. ROOTNEIGHBORS consists of trees of depth 2 with random features of dimension 5. The generation (Appendix E.3) ensures each tree root has at least one degree-6 neighbor. An example is shown on the left of Figure 3: the root node r has only two neighbors with degree 6 (u and v) and the target prediction value is $(\boldsymbol{x}_u + \boldsymbol{x}_v)/2$.

Setup. We consider GCN, GAT, SUMGNN, MEANGNN, as baselines, and compare to CO-GNN(Σ, Σ), CO-GNN(μ, μ), CO-GNN(Σ, α) and CO-GNN(Σ, μ). We report the Mean Average Error (MAE), use the Adam optimizer and present all details including the hyperparameters in Appendix E.4.

Results for MPNNs. The results are presented in Table 1, which includes the random baseline (i.e., MAE obtained via a random prediction). All MPNNs perform poorly: GCN, GAT, and MEANGNN fail to identify node degrees, making it impossible to detect nodes with a specific degree, which is crucial for the task. GCN and GAT are only marginally better than the random baseline, whereas MEANGNN performs substantially better than the random baseline. The latter can be explained by the fact that MEANGNN employs a different transformation on the source node rather than treating it as a neighbor (unlike the self-loop in GCN/GAT). SUMGNN uses sum aggregation and can identify the node degrees, but struggles in averaging the node features, which yields comparable MAE results to that of MEANGNN.

Table 1: Results on ROOTNEIGHBORS. Top three models are colored by First, Second, Third.

Model	MAE
Random	0.474
GAT	0.442
SUMGNN	0.370
MEANGNN	0.329
CO-GNN(Σ, Σ)	0.196
CO-GNN(μ, μ)	0.339
CO-GNN(Σ, α)	0.085
CO-GNN(Σ, μ)	0.079

Results for CO-GNNs. The ideal mode of operation for CO-GNNs would be as follows:

1. The action network chooses either LISTEN or STANDARD for the root node, and BROADCAST or STANDARD for the root-neighbors which have a degree 6.
2. The action network chooses either LISTEN or ISOLATE for all the remaining root-neighbors.
3. The environment network updates the root node by averaging features from its broadcasting neighbors.

CO-GNN(Σ, μ): The best result is achieved by this model, because SUMGNN as the action network can accomplish (1) and (2), and MEANGNN as the environment network can accomplish (3). Therefore, this model leverages the strengths of SUMGNN and MEANGNN to cater to the different roles of the action and environment networks, making it the most natural CO-GNN model for the regression task.

CO-GNN(Σ, α): We observe a very similar phenomenon here to that of CO-GNN(Σ, μ). The action network allows GAT to determine the right topology, and GAT only needs to learn to average the features. This shows the contribution of CO-GNNs is orthogonal to that of attention aggregation.

CO-GNN(Σ, Σ): This model also performs well, primarily because it uses SUMGNN as the action network, accomplishing (1) and (2). However, it uses another SUMGNN as the environment network which cannot easily mimic the averaging of the neighbor's features.

Table 2: Results on node classification. Top three models are colored by First, Second, Third.

	roman-empire	amazon-ratings	minesweeper	tolokers	questions
GCN	73.69 ± 0.74	48.70 ± 0.63	89.75 ± 0.52	83.64 ± 0.67	76.09 ± 1.27
SAGE	85.74 ± 0.67	53.63 ± 0.39	93.51 ± 0.57	82.43 ± 0.44	76.44 ± 0.62
GAT	80.87 ± 0.30	49.09 ± 0.63	92.01 ± 0.68	83.70 ± 0.47	77.43 ± 1.20
GAT-sep	88.75 ± 0.41	52.70 ± 0.62	93.91 ± 0.35	83.78 ± 0.43	76.79 ± 0.71
GT	86.51 ± 0.73	51.17 ± 0.66	91.85 ± 0.76	83.23 ± 0.64	77.95 ± 0.68
GT-sep	87.32 ± 0.39	52.18 ± 0.80	92.29 ± 0.47	82.52 ± 0.92	78.05 ± 0.93
Co-GNN(Σ, Σ)	91.57 ± 0.32	51.28 ± 0.56	95.09 ± 1.18	83.36 ± 0.89	80.02 ± 0.86
Co-GNN(μ, μ)	91.37 ± 0.35	54.17 ± 0.37	97.31 ± 0.41	84.45 ± 1.17	76.54 ± 0.95

Co-GNN(μ, μ): This model clearly performs weakly, since MeanGNN as an action network cannot achieve (1) hindering the performance of the whole task. Indeed, Co-GNN(μ, μ) performs comparably to MeanGNN suggesting that the action network is not useful in this case.

To shed light on the performance of Co-GNN models, we computed the percentage of edges which are accurately retained or removed by the action network in a single layer Co-GNN model. We observe an accuracy of 99.71% for Co-GNN(Σ, μ), 99.55% for Co-GNN(Σ, Σ), and 57.20% for Co-GNN(μ, μ). This empirically confirms the expected behavior of Co-GNNs. In fact, the example tree is shown on the right of Figure 3 is taken from the experiment with Co-GNN(Σ, μ): reassuringly, this model learns precisely the actions that induce the shown optimal subgraph.

6.2. Node Classification with Heterophilic Graphs

One of the strengths of Co-GNNs is their capability to utilize task-specific information propagation, which raises an obvious question: *could* Co-GNN*s outperform the baselines on heterophilious graphs, where standard message passing is known to suffer?* To answer this question, we assess the performance of Co-GNNs on heterophilic node classification datasets from (Platonov et al., 2023).

Setup. We evaluate SumGNN, MeanGNN and their Co-GNN counterparts, Co-GNN(Σ, Σ) and Co-GNN(μ, μ) on the 5 heterophilic graphs, following the 10 data splits and the methodology of Platonov et al. (2023). We report the accuracy and standard deviation for roman-empire and amazon-ratings. We also report the ROC AUC and standard deviation for minesweeper, tolokers, and questions. The classical baselines GCN, GraphSAGE, GAT, GAT-sep, GT (Shi et al., 2021) and GT-sep are taken from Platonov et al. (2023). We use the Adam optimizer and report all hyperparameters in Appendix E.4.

Results. All results are reported in Table 2. Observe that Co-GNNs achieve state-of-the-art results across the board, despite using relatively simple architectures as their action and environment networks. Importantly, Co-GNNs demonstrate an average accuracy improvement of 2.23% compared to all baseline methods, across all datasets, surpassing the performance of more complex models such as GT. In our main finding we observe a consistent trend: enhancing standard models with action networks of Co-GNNs results in improvements in performance. For example, we report 3.19% improvement in accuracy on the roman-empire and 3.62% improvement in ROC AUC on minesweeper compared to the best performing baseline. This shows that Co-GNNs are flexible and effective on different datasets and tasks. These results are reassuring as they establish Co-GNNs as a strong method in the heterophilic setting due to its unique ability to manipulate information flow.

7. Empirical Insights for the Actions

The action network of Co-GNNs is the key model component. The purpose of this section is to provide additional insights regarding the actions being learned by Co-GNNs.

7.1. Actions on Heterophilic vs Homophilic Graphs

We aim to compare the actions learned on a homophilic task to the actions learned on a heterophilic task. One idea would be to inspect the learned action distributions, but they alone may not provide a clear picture of the graph's topology. For example, two connected nodes that choose to Isolate achieve the same topology as nodes that choose both to Broadcast or Listen. This is a result of the immense number of action configurations and their interactions.

To better understand the learned graph topology, we inspect the *induced directed graphs* at every layer. Specifically, we present the *ratio of the directed edges that are kept* across the different layers in Figure 4. We record the directed edge ratio over the 10 different layers of our best, fully trained 10 Co-GNN(μ, μ) models on the roman-empire (Platonov et al., 2023) and cora datasets (Pei et al., 2020). We follow the 10 data splits and the methodology of Platonov et al. (2023) and Yang et al. (2016), respectively.

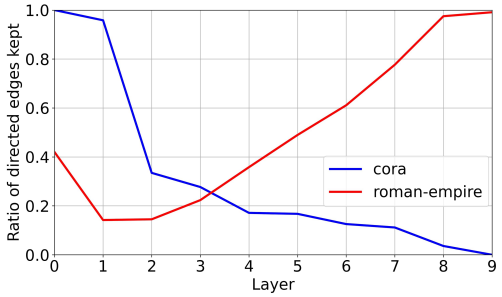


Figure 4: The ratio of directed edges that are kept on cora (as a homophilic dataset) and on roman-empire (as a heterophilic dataset) for each layer $0 \leq \ell < 10$.

This experiment serves as a strong evidence for the adaptive nature of CO-GNNs this statement. Indeed, by inspecting Figure 4, we observe completely *opposite trends* between the two datasets.

On the homophilic dataset cora, the ratio of edges that are kept gradually *decreases* as we go to the deeper layers. In fact, 100% of the edges are kept at layer $\ell = 0$ while *all* edges are dropped at layer $\ell = 9$. This is very insightful because homophilic datasets are known to *not* benefit from using many layers, and the trained CO-GNN model recognizes this by eventually isolating all the nodes. This is particularly the case for cora, where classical MPNNs typically achieve their best performance with 1-3 layers.

On the heterophilic dataset roman-empire, the ratio of edges that are kept gradually *increases* after $\ell = 1$ as we go to the deeper layers. Initially, $\sim 42\%$ of the edges are kept at layer $\ell = 0$ while eventually this reaches 99% at layer $\ell = 9$. This is interesting, since in heterophilic graphs, edges tend to connect nodes of different classes and so classical MPNNs, which aggregate information based on the homophily assumption perform poorly. Although, CO-GNN model uses these models it compensates by controlling information flow. The model manages to capture the heterophilous aspect of the dataset by restricting the flow of information in the early layer and slowly enabling it the deeper the layer (the further away the nodes), which might be a great benefit to its success over heterophilic benchmarks.

7.2. What Actions are Performed on Minesweeper?

To better understand the topology learned by CO-GNNs, we visualize the topology at each layer in a CO-GNN model over the highly regular minesweepers dataset.

Dataset. Minesweeper (Platonov et al., 2023) is a synthetic dataset inspired by the Minesweeper game. It is a semi-supervised node classification dataset with a regular 100×100 grid where each node is connected to eight neighboring nodes. Each node has an one-hot-encoded input feature

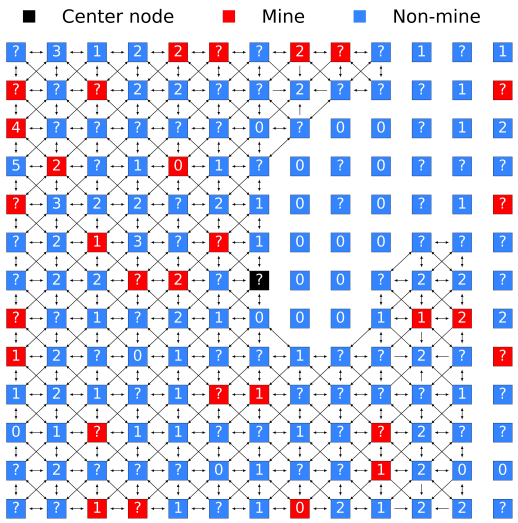


Figure 5: The 10-hop neighborhood at layer $\ell = 4$

showing the number of adjacent mines. A randomly chosen 50% of the nodes have an unknown feature, indicated by a separate binary feature. The task is to identify whether the querying node is a mine.

Setup. We train a 10-layer CO-GNN(μ, μ) model and present the graph topology at layer $\ell = 4$. The evolution of the graph topology from layer $\ell = 1$ to layer $\ell = 8$ is presented in Appendix F. We choose a node (black), and at every layer ℓ, we depict its neighbors up to distance 10. In this visualization, nodes which are mines are shown in red, and other nodes in blue. The features of non-mine nodes (indicating the number of neighboring mines) are shown explicitly whereas the nodes whose features are hidden are labeled with a question mark.

Interpreting the Actions. The visualization of the actions at layer $\ell = 4$ is shown in Figure 5. In this game, every label is informative: non-0-labeled nodes are more informative in the earlier layers ($\ell = 1, 2, 3, 4$), whereas 0-labeled nodes become more informative at later layers. This can be explained by considering two cases:

1. The target node has *at least one* 0-labeled neighbor: In this case, the prediction trivializes, since we know that the target node cannot be a mine.
2. The target node has *no* 0-labeled neighbors: In this case, the model needs to make a sophisticated inference based on the surrounding mines within the k-hop distance. In this scenario, a node obtains more information by aggregating from non-0-label nodes. The model can still implicitly infer "no mine" from the lack of a signal/message.

The action network appears to diffuse the information of non-0-labeled nodes in the early layers to capture nodes

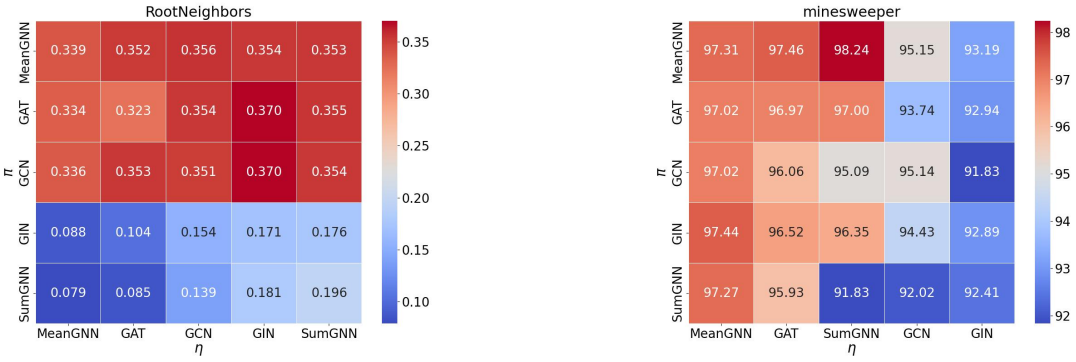


Figure 6: MAE and ROC AUC as a function of the choice of action (π) and environment (η) networks over ROOTNEIGHBORS (**left**) and the minesweeper (**right**) experiments, respectively.

from case (2), while largely isolating the 0-labeled nodes until the last layers (avoiding mixing of signals and potential loss of information) which can later be used to capture the nodes from case (1). In the earlier layers, the action network prioritizes the information flowing from the left sections of the grid in Figure 5 where more mines are present. After identifying the most crucial information and propagating this through the network, it then requires this information to also be communicated with the nodes that initially were labeled with 0. This leads to an almost fully connected grid in the later layers (see $\ell = 7, 8$ in Figures 16 and 17).

7.3. Which Action or Environment Network?

We conduct an ablation study on the choice of action and environment networks to quantify its affect.

Setup. We experiment with all 25 combinations of the architectures MEANGNNs, GAT, SUMGNN, GCN and GIN on the heterophilic graph minesweeper and on the synthetic dataset ROOTNEIGHBORS. We report MAE for ROOTNEIGHBORS and the ROC AUC for minesweeper.

ROOTNEIGHBORS Results. The results reported in Figure 6 (left) support our analysis from Section 6.1: an environment network with mean-type aggregation (GCN, GAT, or MEANGNN) and an action network with sum-type aggregation (SUMGNN, or GIN) are best choices for the task. The choice of the action network is critical for this task: SUMGNN and GIN yield best results across the board when they are used as action networks. In contrast, if we use GAT, GCN, or MEANGNN as action networks then the results are poor and comparable to baseline results on this task. These action networks cannot detect node cardinality which prevents them from choosing the optimal actions as elaborated in Section 6.1. The choice of the environment network is relatively less important for this task.

Minesweeper Results. In Figure 6 (right), CO-GNNs achieve multiple state-of-the-art results on minesweeper with different choices of action and environment networks.

In terms of the environment network, we observe that MeanGNN and GAT yield consistently robust results when used as environment networks regardless of the choice of the action network. This makes sense in the context of the minesweeper game. In order to make a good inference, a k-layer environment network can keep track of the average number of mines found in each hop distance. The task is hence well-suited to mean-style aggregation environment networks, which manifests as empirical robustness. GCN performs worse as environment network, since it cannot distinguish a node from its neighbors.

In terms of the action network, we observed earlier that the role of the action network is to mainly make a distinction between 0-labeled nodes and non-0-labeled nodes, and such an action network can be realized with all of the architectures considered. As a result, we do not observe dramatic differences in the performance regarding to the choice of the action network in this task.

8. Summary and Outlook

We introduced CO-GNN architectures which can dynamically explore the graph topology while learning. These architectures have desirable properties which can inform future work. Looking forward, one potential future direction is to adapt CO-GNNs to other types of graphs such as directed, and even multi-relational graphs. One possible approach is by including actions that also consider the directionality. For example, for each node u, one can define the actions LISTEN-INC (listen to nodes that have an incoming edge to u) and LISTEN-OUT (listen to nodes that have an incoming edge from u) and extend the other actions analogously to incorporate directionality. It is also possible to consider other types of actions or extend our approach to edge-wise actions (rather than node-wise), though these extensions will lead to a larger state space.

Acknowledgments

The authors would like to thank the anonymous reviewers for their feedback which led to substantial improvements in the presentation of the paper. The first author is funded by the Clarendon scholarship. The authors would like to also acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work (http://dx.doi.org/10.5281/zenodo.22558). This work was also partially funded by EPSRC Turing AI World-Leading Research Fellowship No. EP/X040062/1.

Impact Statement

This paper presents a novel graph neural network paradigm whose goal is to explore the graph topology while learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Abboud, R., Ceylan, İ. İ., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. In *IJCAI*, 2021.

Abboud, R., Dimitrov, R., and Ceylan, İ. İ. Shortest path networks for graph property prediction. In *LoG*, 2022.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *ICLR*, 2021.

Bacciu, D., Errica, F., and Micheli, A. Probabilistic learning on graphs via contextual architectures. In *JMLR*, 2020.

Barceló, P., Geerts, F., Reutter, J. L., and Ryschkov, M. Graph neural networks with local graph parameters. In *NeurIPS*, 2021.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. In *ICLR*, 2022.

Bodnar, C., Giovanni, F. D., Chamberlain, B. P., Liò, P., and Bronstein, M. M. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in GNNs. In *NeurIPS*, 2023.

Bresson, X. and Laurent, T. Residual gated graph convnets. In *arXiv*, 2018.

Castellana, D., Errica, F., Bacciu, D., and Micheli, A. The infinite contextual graph Markov model. In *ICML*, 2022.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *ICML*, 2020.

Cybenko, G. Approximation by superpositions of a sigmoidal function. In *MCSS*, 1989.

Dai, E., Jin, W., Liu, H., and Wang, S. Towards robust graph neural networks for noisy graphs with sparse labels. In *WSDM*, 2022.

Di Giovanni, F., Giusti, L., Barbero, F., Luise, G., Lio, P., and Bronstein, M. M. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *ICLR*, 2023.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, 2015.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. In *NeurIPS Datasets and Benchmarks*, 2022.

Errica, F. and Niepert, M. Tractable probabilistic graph representation learning with graph-induced sum-product networks. In *arXiv*, 2023.

Errica, F., Podda, M., Bacciu, D., and Micheli, A. A fair comparison of graph neural networks for graph classification. In *ICLR*, 2020.

Faber, L. and Wattenhofer, R. Asynchronous neural networks for learning in graphs. In *arXiv*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, 2017.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *IJCNN*, 2005.

Gutteridge, B., Dong, X., Bronstein, M. M., and Di Giovanni, F. DRew: Dynamically rewired message passing with delay. In *ICML*, 2023.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

Hamilton, W. L. Graph representation learning. In *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 2020.

He, X., Hooi, B., Laurent, T., Perold, A., Lecun, Y., and Bresson, X. A generalization of ViT/MLP-mixer to graphs. In *ICML*, 2023.

Hornik, K. Approximation capabilities of multilayer feedforward networks. In *Neural Networks*, 1991.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Jin, E., Bronstein, M., Ceylan, İ. İ., and Lanzinger, M. Homomorphism counts for graph neural networks: All about that basis. In *ICML*, 2024.

Karhadkar, K., Banerjee, P. K., and Montufar, G. FoSR: First-order spectral rewiring for addressing oversquashing in GNNs. In *ICLR*, 2023.

Keriven, N. and Peyré, G. Universal invariant and equivariant graph neural networks. In *NeurIPS*, 2019.

Kipf, T. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Klicpera, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *NeurIPS*, 2019.

Lai, K.-H., Zha, D., Zhou, K., and Hu, X. Policy-GNN: Aggregation optimization for graph neural networks. In *KDD*, 2020.

Li, Q., Han, Z., and Wu, X. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

Loukas, A. What graph neural networks cannot learn: depth vs width. In *ICLR*, 2020.

Ma, L., Lin, C., Lim, D., Romero-Soriano, A., Dokania, K., Coates, M., H.S. Torr, P., and Lim, S.-N. Graph Inductive Biases in Transformers without Message Passing. In *ICML*, 2023.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. In *NeurIPS*, 2019.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML workshop on Graph Representation Learning and Beyond*, 2020.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-GCN: Geometric graph convolutional networks. In *ICLR*, 2020.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *ICLR*, 2023.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. In *SDM*, 2021.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. In *IEEE Transactions on Neural Networks*, 2009.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. In *AI Magazine*, 2008.

Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. Masked label prediction: Unified message passing model for semi-supervised classification. In *IJCAI*, 2021.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., and Sinop, A. K. Exphormer: Sparse transformers for graphs. In *ICML*, 2023.

Shlomi, J., Battaglia, P., and Vlimant, J.-R. Graph neural networks in particle physics. In *MLST*, 2021.

Simonovsky, M. and Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*, 2017.

Thiede, E. H., Zhou, W., and Kondor, R. Autobahn: Automorphism-based graph neural nets. In *NeurIPS*, 2021.

Tönshoff, J., Ritzert, M., Wolf, H., and Grohe, M. Walking out of the weisfeiler leman hierarchy: Graph learning beyond message passing. In *TMLR*, 2023.

Topping, J., Giovanni, F. D., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*, 2022.

Tönshoff, J., Ritzert, M., Rosenbluth, E., and Grohe, M. Where did the gap go? reassessing the long-range graph benchmark. In *arXiv*, 2023.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *ICLR*, 2018.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph CNN for learning on point clouds. In *TOG*, 2019.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. In *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2019.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T. Do transformers really perform badly for graph representation? In *NeurIPS*, 2021.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. *arXiv*, 2019.

Zitnik, M., Agrawal, M., and Leskovec, J. Modeling polypharmacy side effects with graph convolutional networks. In *Bioinformatics*, 2018.

A. Proofs of Technical Results

A.1. Proof of Proposition 5.1

In order to prove Proposition 5.1, we first prove the following lemma, which shows that all non-isolated nodes of an input graph can be individualized by CO-GNNs:

Lemma A.1. *Let $G = (V, E, \boldsymbol{X})$ be a graph with node features. For every pair of non-isolated nodes $u, v \in V$ and for all $\delta > 0$, there exists a* CO-GNN *architecture with sufficiently many layers L which satisfies $\mathbb{P}(\boldsymbol{h}_u^{(L)} \neq \boldsymbol{h}_v^{(L)}) \geq 1 - \delta$.*

Proof. We consider an L-layer CO-GNN(η, π) architecture satisfying the following:

(i) the *environment* network η is composed of L injective layers,

(ii) the *action* network π is composed of a *single* layer, and it is shared across CO-GNN layers.

Item (i) can be satisfied by a large class of GNN architectures, including SUMGNN (Morris et al., 2019) and GIN (Xu et al., 2019). We start by assuming $\boldsymbol{h}_u^{(0)} = \boldsymbol{h}_v^{(0)}$. These representations can be differentiated if the model can jointly realize the following actions at some layer ℓ using the action network π:

1. $a_u^{(\ell)} = \mathrm{L} \vee \mathrm{S}$,
2. $a_v^{(\ell)} = \mathrm{I} \vee \mathrm{B}$, and
3. $\exists$ a neighbor w of u s.t. $a_w^{(\ell)} = \mathrm{S} \vee \mathrm{B}$.

The key point is to ensure an update for the state of u via an aggregated message from its neighbors (at least one), while isolating v. In what follows, we assume the worst-case for the degree of u and consider a node w to be the only neighbor of u. Let us denote the joint probability of realizing these actions 1-3 for the nodes u, v, w at layer ℓ as:

$$p_{u,v}^{(\ell)} = \mathbb{P}\left((a_u^{(\ell)} = \mathrm{L} \vee \mathrm{S}) \wedge (a_v^{(\ell)} = \mathrm{I} \vee \mathrm{B}) \wedge (a_w^{(\ell)} = \mathrm{S} \vee \mathrm{B}) \right).$$

The probability of taking each action is non-zero (since it is a result of applying softmax) and u has at least one neighbor (non-isolated), therefore $p_{u,v}^{(\ell)} > 0$. For example, if we assume a constant action network that outputs a uniform distribution over the possible actions (each action probability 0.25) then $p_{u,v}^{(\ell)} = 0.125$.

This means that the environment network η applies the following updates to the states of u and v with probability $p_{u,v}^{(\ell)} > 0$:

$$\boldsymbol{h}_u^{(\ell+1)} = \eta^{(\ell)}\left(\boldsymbol{h}_u^{(\ell)}, \{\!\{\boldsymbol{h}_w^{(\ell)} \mid w \in \mathcal{N}_u, a_w^{(\ell)} = \mathrm{S} \vee \mathrm{B}\}\!\}\right),$$
$$\boldsymbol{h}_v^{(\ell+1)} = \eta^{(\ell)}\left(\boldsymbol{h}_v^{(\ell)}, \{\!\{\}\!\}\right).$$

The inputs to the environment network layer $\eta^{(\ell)}$ for these updates are clearly different, and since the environment layer is injective, we conclude that $\boldsymbol{h}_u^{(\ell+1)} \neq \boldsymbol{h}_v^{(\ell+1)}$.

Thus, the probability of having different final representations for the nodes u and v is lower bounded by the probability of the events 1-3 jointly occurring at least once in one of the CO-GNN layers, which, by applying the union bound, yields:

$$\mathbb{P}(\boldsymbol{h}_u^{(L_{u,v})} \neq \boldsymbol{h}_v^{(L_{u,v})}) \geq 1 - \prod_{\ell=0}^{L_{u,v}} \left(1 - p_{u,v}^{(\ell)}\right) \geq 1 - (1 - \gamma_{u,v})^{L_{u,v}} \geq 1 - \delta$$

where $\gamma_{u,v} = \max_{\ell \in [L_{u,v}]} \left(p_{u,v}^{(\ell)}\right)$ and $L_{u,v} = \log_{1-\gamma_{u,v}}(\delta)$.

We repeat this process for all pairs of non-isolated nodes $u, v \in V$. Due to the injectivity of $\eta^{(\ell)}$ for all $\ell \in [L]$, once the nodes are distinguished, they cannot remain so in deeper layers of the architecture, which ensures that all nodes

$u, v \in V$ differ in their final representations $\boldsymbol{h}_u^{(\ell)} \neq \boldsymbol{h}_v^{(\ell)}$ after this process completes. The number of layers required for this construction is then given by:

$$L = |V \setminus I| \log_{1-\alpha}(\delta) \geq \sum_{u,v \in V \setminus I} \log_{1-\gamma_{u,v}}(\delta),$$

where I is the set of all isolated nodes in V and

$$\alpha = \max_{u,v \in V \setminus I} (\gamma_{u,v}) = \max_{u,v \in V \setminus I} \left(\max_{\ell \in [L_{u,v}]} \left(p_{u,v}^{(\ell)} \right) \right).$$

Having shown a CO-GNN construction with the number of layers bounded as above, we conclude the proof. □

Proposition 5.1. *Let $G_1 = (V_1, E_1, \boldsymbol{X}_1)$ and $G_2 = (V_2, E_2, \boldsymbol{X}_2)$ be two non-isomorphic graphs. Then, for any threshold $0 < \delta < 1$, there exists a parametrization of a* CO-GNN *architecture using sufficiently many layers L, satisfying $\mathbb{P}(\boldsymbol{z}_{G_1}^{(L)} \neq \boldsymbol{z}_{G_2}^{(L)}) \geq 1 - \delta$.*

Proof. Let $\delta > 0$ be any value and consider the graph $G = (V, E, \boldsymbol{X})$ which has G_1 and G_2 as its components:

$$V = V_1 \cup V_2, \quad E = E_1 \cup E_2, \quad \boldsymbol{X} = \boldsymbol{X}_1 || \boldsymbol{X}_2,$$

where $||$ is the matrix horizontal concatenation. By Lemma A.1, for every pair of non-isolated nodes $u, v \in V$ and for all $\delta > 0$, there exists a CO-GNN architecture with sufficiently many layers $L = |V \setminus I| \log_{1-\alpha}(\delta)$ which satisfies:

$$\mathbb{P}(\boldsymbol{h}_u^{(L_{u,v})} \neq \boldsymbol{h}_v^{(L_{u,v})}) \geq 1 - \delta, \text{ with } \alpha = \max_{u,v \in V \setminus I} \left(\max_{\ell \in [L_{u,v}]} \left(p_{u,v}^{(\ell)} \right) \right),$$

where $p_{u,v}^{(\ell)}$ represents a lower bound on the probability for the representations of nodes $u, v \in V$ at layer ℓ being different.

We use the same CO-GNN construction given in Lemma A.1 on G, which ensures that all non-isolated nodes have different representations in G. When applying this CO-GNN to G_1 and G_2 separately, we get that every non-isolated node from either graph has a different representation with probability $1 - \delta$ as a result. Hence, the multiset $\mathcal{M}_1$ of node features for G_1 and the multiset $\mathcal{M}_2$ of node features of G_2 must differ. Assuming an injective pooling function from these multisets to graph-level representations, we get:

$$\mathbb{P}(\boldsymbol{z}_{G_1}^{(L)} \neq \boldsymbol{z}_{G_2}^{(L)}) \geq 1 - \delta$$

for $L = |V \setminus I| \log_{1-\alpha}(\delta)$. □

A.2. Proof of Proposition 5.2

Proposition 5.2. *Let $G = (V, E, \boldsymbol{X})$ be a connected graph with node features. For some $k > 0$, for any target node $v \in V$, for any k source nodes $u_1, \ldots, u_k \in V$, and for any compact, differentiable function $f : \mathbb{R}^{d^{(0)}} \times \ldots \times \mathbb{R}^{d^{(0)}} \to \mathbb{R}^d$, there exists an L-layer* CO-GNN *computing final node representations such that for any $\epsilon, \delta > 0$ it holds that $\mathbb{P}(|\boldsymbol{h}_v^{(L)} - f(\boldsymbol{x}_{u_1}, \ldots \boldsymbol{x}_{u_k})| < \epsilon) \geq 1 - \delta$.*

Proof. For arbitrary $\epsilon, \delta > 0$, we start by constructing a feature encoder $\text{ENC} : \mathbb{R}^{d^{(0)}} \to \mathbb{R}^{2(k+1)d^{(0)}}$ which encodes the initial representations $\boldsymbol{x}_w \in \mathbb{R}^{d^{(0)}}$ of each node w as follows:

$$\text{ENC}(\boldsymbol{x}_w) = [\underbrace{\tilde{\boldsymbol{x}}_w^\top \oplus \ldots \oplus \tilde{\boldsymbol{x}}_w^\top}_{k+1}]^\top,$$

where $\tilde{\boldsymbol{x}}_w = [\text{ReLU}(\boldsymbol{x}_w^\top) \oplus \text{ReLU}(-\boldsymbol{x}_w^\top)]^\top$. Observe that this encoder can be parametrized using a 2-layer MLP, and that $\tilde{\boldsymbol{x}}_w$ can be decoded using a single linear layer to get back to the initial features:

$$\text{DEC}(\tilde{\boldsymbol{x}}_w) = \text{DEC}\left([\text{ReLU}(\boldsymbol{x}_w^\top) \oplus \text{ReLU}(-\boldsymbol{x}_w^\top)]^\top \right) = \boldsymbol{x}_w$$

Individualizing the Graph. Importantly, we encode the features using $2(k+1)d^{(0)}$ dimensions in order to be able to preserve the original node features. Using the construction from Lemma A.1, we can ensure that every pair of nodes in the

connected graph have different features with probability $1-\delta_1$. However, if we do this naïvely, then the node features will be changed before we can transmit them to the target node. We therefore make sure that the width of the CO-GNN architecture from Lemma A.1 is increased to $2(k+1)d^{(0)}$ dimensions such that it applies the identity mapping on all features beyond the first $2d^{(0)}$ components. This way we make sure that all feature components beyond the first $2d^{(0)}$ components are preserved. The existence of such a CO-GNN is straightforward since we can always do an identity mapping using base environment models such SUMGNNs. We use L_1 CO-GNN layers for this part of the construction.

In order for our architecture to retain a positive representation for all nodes, we now construct 2 additional layers which encode the representation $\boldsymbol{h}_w^{(L)} \in \mathbb{R}^{2(k+1)d^{(0)}}$ of each node w as follows:

$$[\mathrm{ReLU}(\boldsymbol{q}_w^\top) \oplus \mathrm{ReLU}(-\boldsymbol{q}_w^\top) \oplus \tilde{\boldsymbol{x}}_w^\top \oplus \ldots \oplus \tilde{\boldsymbol{x}}_w^\top]^\top$$

where $\boldsymbol{q}_w \in \mathbb{R}^{2d^{(0)}}$ denotes a vector of the first $2d^{(0)}$ entries of $\boldsymbol{h}_w^{(L_1)}$.

Transmitting Information. Consider a shortest path $u_1 = w_0 \to w_1 \to \cdots \to w_r \to w_{r+1} = v$ of length r_1 from node u_1 to node v. We use exactly r_1 CO-GNN layers in this part of the construction. For the first these layers, the action network assigns the following actions to these nodes:

- w_0 performs the action BROADCAST,
- w_1 performs the action LISTEN, and
- all other nodes are perform the action ISOLATE.

This is then repeated in the remaining layers, for all consecutive pairs w_i, w_{i+1}, $0 \leq i \leq r$ until the whole path is traversed. That is, at every layer, all graph edges are removed except the one between w_i and w_{i+1}, for each $0 \leq i \leq r$. By construction each element in the node representations is positive and so we can ignore the ReLU.

We apply the former construction such that it acts on entries $2d^{(0)}$ to $3d^{(0)}$ of the node representations, resulting in the following representation for node v:

$$[\mathrm{ReLU}(\boldsymbol{q}_w^\top) \oplus \mathrm{ReLU}(-\boldsymbol{q}_w^\top) \oplus \tilde{\boldsymbol{x}}_{u_1} \oplus \tilde{\boldsymbol{x}}_v \oplus \ldots \oplus \tilde{\boldsymbol{x}}_v]$$

where $\boldsymbol{q}_w \in \mathbb{R}^{2d^{(0)}}$ denotes a vector of the first $2d^{(0)}$ entries of $\boldsymbol{h}_w^{(L_1)}$.

We denote the probability in which node y does not follow the construction at stage $1 \leq t \leq r$ by $\beta_y^{(t)}$ such that the probability that all graph edges are removed except the one between w_i and w_{i+1} at stage t is lower bounded by $(1-\beta)^{|V|}$, where $\beta = \max_{y \in V}(\beta_y)$. Thus, the probability that the construction holds is bounded by $(1-\beta)^{|V|r_1}$.

The same process is then repeated for nodes u_i, $2 \leq i \leq k$, acting on the entries $(k+1)d^{(0)}$ to $(k+2)d^{(0)}$ of the node representations and resulting in the following representation for node v:

$$[\mathrm{ReLU}(\boldsymbol{q}_w^\top) \oplus \mathrm{ReLU}(-\boldsymbol{q}_w^\top) \oplus \tilde{\boldsymbol{x}}_{u_1} \oplus \tilde{\boldsymbol{x}}_{u_2} \oplus \ldots \oplus \tilde{\boldsymbol{x}}_{u_k}]$$

In order to decode the positive features, we construct the feature decoder $\mathrm{DEC}' : \mathbb{R}^{2(k+2)d^{(0)}} \to \mathbb{R}^{(k+1)d^{(0)}}$, that for $1 \leq i \leq k$ applies DEC to entries $2(i+1)d^{(0)}$ to $(i+2)d^{(0)}$ of its input as follows:

$$[\mathrm{DEC}(\tilde{\boldsymbol{x}}_{u_1}) \oplus \ldots \oplus \mathrm{DEC}(\tilde{\boldsymbol{x}}_{u_k})] = [\boldsymbol{x}_{u_1} \oplus \ldots \oplus \boldsymbol{x}_{u_k}]$$

Given ϵ, δ, we set:

$$\delta_2 = 1 - \frac{1-\delta}{(1-\delta_1)\,(1-\beta)^{|V|\sum_{i=1}^{k+1} r_i}} > 0.$$

Having transmitted and decoded all the required features into $\boldsymbol{x} = [\boldsymbol{x}_1 \oplus \ldots \oplus \boldsymbol{x}_k]$, where $\boldsymbol{x}_i$ denotes the vector of entries $id^{(0)}$ to $(i+1)d^{(0)}$ for $0 \leq i \leq k$, we can now use an $\mathrm{MLP} : \mathbb{R}^{(k+1)d^{(0)}} \to \mathbb{R}^d$ and the universal approximation property to map this vector to the final representation $\boldsymbol{h}_v^{(L)}$ such that:

$$\mathbb{P}(|\boldsymbol{h}_v^{(L)} - f(\boldsymbol{x}_{u_1}, \ldots \boldsymbol{x}_{u_k})| < \epsilon) \geq (1-\delta_1)\,(1-\beta)^{|V|\sum_{i=1}^{k+1} r_i}\,(1-\delta_2) \geq 1-\delta.$$

The construction hence requires $\left(L = L_1 + 2 + \sum_{i=0}^{k} r_i\right)$ CO-GNN layers. □

B. Relation to Over-squashing

Over-squashing refers to the failure of message passing to propagate information on the graph. Topping et al. (2022) and Di Giovanni et al. (2023) formalized over-squashing as the insensitivity of an r-layer MPNN output at node u to the input features of a distant node v, expressed through a bound on the Jacobian $\|\partial \boldsymbol{h}_v^{(r)}/\partial \boldsymbol{x}_u\| \leq C^r(\hat{\boldsymbol{A}}^r)_{vu}$, where C encapsulated architecture-related constants (e.g., width, smoothness of the activation function, etc.) and the normalized adjacency matrix $\hat{\boldsymbol{A}}$ captures the effect of the graph. Graph rewiring techniques amount to modifying $\hat{\boldsymbol{A}}$ so as to increase the upper bound and thereby reduce the effect of over-squashing.

Observe that the actions of every node in Co-GNNs result in an effective graph rewiring (different at every layer). As a result, the action network can choose actions that transmit the features of node $u \in V$ to node $v \in V$ as shown in Proposition 5.2, resulting in the maximization of the bound on the Jacobian between a pair of nodes or (k nodes, for some fixed k).

C. Additional Experiments

C.1. Expressivity Experiment

In Proposition 5.1 we state that Co-GNNs can distinguish between pairs of graphs which are 1-WL indistinguishable. We validate this with a simple synthetic dataset: Cycles. Cycles consists of 7 pairs of undirected graphs, where the first graph is a k-cycle for $k \in [6, 12]$ and the second graph is a disjoint union of a $(k-3)$-cycle and a triangle. The train/validation/test set are the $k \in [6, 7]/[8, 9]/[10, 12]$ pairs, correspondingly. The task is to correctly identify the cycle graphs. As the pairs are 1-WL indistinguishable, solving this task implies a strictly higher expressive power than 1-WL.

Our main finding is that Co-GNN(Σ, Σ) and Co-GNN(μ, μ) achieve 100% accuracy, perfectly classifying the cycles, whereas their corresponding classical SumGNN and MeanGNN achieve a random guess accuracy of 50%. These results imply that Co-GNN can increase the expressive power of their classical counterparts. We find the model behaviour rather volatile during training, which necessitated careful tuning of hyperparameters.

C.2. Long-range Interactions

To validate the performance of Co-GNNs on long-range tasks, we experiment with the LRGB benchmark (Dwivedi et al., 2022).

Setup. We train Co-GNN$(*, *)$ and Co-GNN(ϵ, ϵ) Co-GNN(ϵ, ϵ) on LRGB and report the unweighted mean Average Precision (AP) for Peptides-func. All experiments are run 4 times with 4 different seeds and follow the data splits provided by Dwivedi et al. (2022). Following the methodology of Tönshoff et al. (2023), we used AdamW as optimizer and cosine-with-warmup scheduler. We also use the provided results for GCN, GCNII (Chen et al., 2020), GINE, GatedGCN (Bresson & Laurent, 2018), CRaWl (Tönshoff et al., 2023), DRew (Gutteridge et al., 2023), Exphormer (Shirzad et al., 2023), GRIT (Ma et al., 2023), Graph-ViT / G-MLPMixer (He et al., 2023).

Table 3: Results on LRGB. Top three models are colored by First, Second, Third.

	Peptides-func
GCN	0.6860 ± 0.0050
GINE	0.6621 ± 0.0067
GatedGCN	0.6765 ± 0.0047
CRaWl	0.7074 ± 0.0032
DRew	0.7150 ± 0.0044
Exphormer	0.6527 ± 0.0043
GRIT	0.6988 ± 0.0082
Graph-ViT	0.6942 ± 0.0075
G-MLPMixer	0.6921 ± 0.0054
Co-GNN$(*, *)$	0.6990 ± 0.0093
Co-GNN(ϵ, ϵ)	0.6963 ± 0.0076

Results. We follow Tönshoff et al. (2023) who identified that the previously reported large performance gaps between classical MPNNs and transformer-based models can be closed by a more extensive tuning of MPNNs. In light of this, we note that the performance gap between different models is not large. Classical MPNNs such as GCN, GINE, and GatedGCN surpass some transformer-based approaches such as Exphormer. Co-GNN$(*, *)$ further improves on the competitive GCN and is the third best performing model after DRew and CRaWl. Similarly, Co-GNN(ϵ, ϵ) closely matches Co-GNN$(*, *)$ and is substantially better than its base architecture GIN. This experiment further suggests that exploring different classes of Co-GNNs is a promising direction, as Co-GNNs typically boost the performance of their underlying base architecture.

Table 4: Results on graph classification. Top three models are colored by First, Second, Third.

	IMDB-B	IMDB-M	REDDIT-B	REDDIT-M	NCI1	PROTEINS	ENZYMES
DGCNN	69.2 ± 3.0	45.6 ± 3.4	87.8 ± 2.5	49.2 ± 1.2	76.4 ± 1.7	72.9 ± 3.5	38.9 ± 5.7
DiffPool	68.4 ± 3.3	45.6 ± 3.4	89.1 ± 1.6	53.8 ± 1.4	76.9 ± 1.9	73.7 ± 3.5	59.5 ± 5.6
ECC	67.7 ± 2.8	43.5 ± 3.1	OOR	OOR	76.2 ± 1.4	72.3 ± 3.4	29.5 ± 8.2
GIN	71.2 ± 3.9	48.5 ± 3.3	89.9 ± 1.9	56.1 ± 1.7	80.0 ± 1.4	73.3 ± 4.0	59.6 ± 4.5
GraphSAGE	68.8 ± 4.5	47.6 ± 3.5	84.3 ± 1.9	50.0 ± 1.3	76.0 ± 1.8	73.0 ± 4.5	58.2 ± 6.0
CGMM	-	-	88.1 ± 1.9	52.4 ± 2.2	76.2 ± 2.0	-	-
ICGMM$_f$	71.8 ± 4.4	49.0 ± 3.8	91.6 ± 2.1	55.6 ± 1.7	76.4 ± 1.4	73.2 ± 3.9	-
SPN($k = 5$)	-	-	-	-	78.6 ± 1.7	74.2 ± 2.7	69.4 ± 6.2
GSPN	-	-	90.5 ± 1.1	55.3 ± 2.0	76.6 ± 1.9	-	-
Co-GNN(Σ, Σ)	70.8 ± 3.3	48.5 ± 4.0	88.6 ± 2.2	53.6 ± 2.3	80.6 ± 1.1	73.1 ± 2.3	65.7 ± 4.9
Co-GNN(μ, μ)	72.2 ± 4.1	49.9 ± 4.5	90.5 ± 1.9	56.3 ± 2.1	79.4 ± 0.7	71.3 ± 2.0	68.3 ± 5.7

C.3. Graph Classification

In this experiment, we evaluate Co-GNNs on the TUDataset (Morris et al., 2020) graph classification benchmark.

Setup. We evaluate Co-GNN(Σ, Σ) and Co-GNN(μ, μ) on the 7 graph classification benchmarks, following the risk assessment protocol of Errica et al. (2020), and report the mean accuracy and standard deviation. The results for the baselines DGCNN (Wang et al., 2019), DiffPool (Ying et al., 2018), Edge-Conditioned Convolution (ECC) (Simonovsky & Komodakis, 2017), GIN, GraphSAGE are from Errica et al. (2020). We also include CGMM (Bacciu et al., 2020), ICGMM$_f$ (Castellana et al., 2022), SPN($k = 5$) (Abboud et al., 2022) and GSPN (Errica & Niepert, 2023) as more recent baselines. OOR (Out of Resources) implies extremely long training time or GPU memory usage. We use Adam optimizer and StepLR learn rate scheduler, and report all hyperparameters in the appendix (Table 13).

Results. Co-GNN models achieve the highest accuracy on three datasets in Table 4 and remain competitive on the other datasets. Co-GNN yield these performance improvements, despite using relatively simple action and environment networks, which is intriguing as Co-GNNs unlock a large design space which includes a large class of model variations.

C.4. Homophilic Node Classification

In this experiment, we evaluate Co-GNNs on the homophilic node classification benchmarks cora and pubmed (Sen et al., 2008).

Setup. We assess MeanGNN, SumGNN and their corresponding Co-GNNs counterparts Co-GNN(μ, μ) and Co-GNN(Σ, Σ) on the homophilic graphs and their 10 fixed splits provided by Pei et al. (2020), where we report the mean accuracy, standard deviation and the accuracy gain due to the application of Co-GNN. We also use the results provided by Bodnar et al. (2023) for the classical baseline: GCN, GraphSAGE, GAT, Geom-GCN (Pei et al., 2020) and GCNII.

Results. Table 5 illustrates a modest performance increase of 1-2% across all datasets when transitioning from SumGNN, MeanGNN, and GCN to their respective Co-GNN counterparts. These datasets are highly homophilic, but Co-GNNs nonetheless show improvements on these datasets (even though, modest) compared to their environment/action network architectures.

Table 5: Results on homophilic datasets. Top three models are colored by First, Second, Third.

	pubmed	cora
MLP	87.16 ± 0.37	75.69 ± 2.00
GCN	88.42 ± 0.50	86.98 ± 1.27
GraphSAGE	88.45 ± 0.50	86.90 ± 1.04
GAT	87.30 ± 1.10	86.33 ± 0.48
Geom-GCN	87.53 ± 0.44	85.35 ± 1.57
GCNII	90.15 ± 0.43	88.37 ± 1.25
SumGNN	88.58 ± 0.57	84.80 ± 1.71
MeanGNN	88.66 ± 0.44	84.50 ± 1.25
Co-GNN(Σ, Σ)	89.39 ± 0.39	86.43 ± 1.28
Co-GNN(μ, μ)	89.60 ± 0.42	86.53 ± 1.20
Co-GNN$(*, *)$	89.51 ± 0.88	87.44 ± 0.85

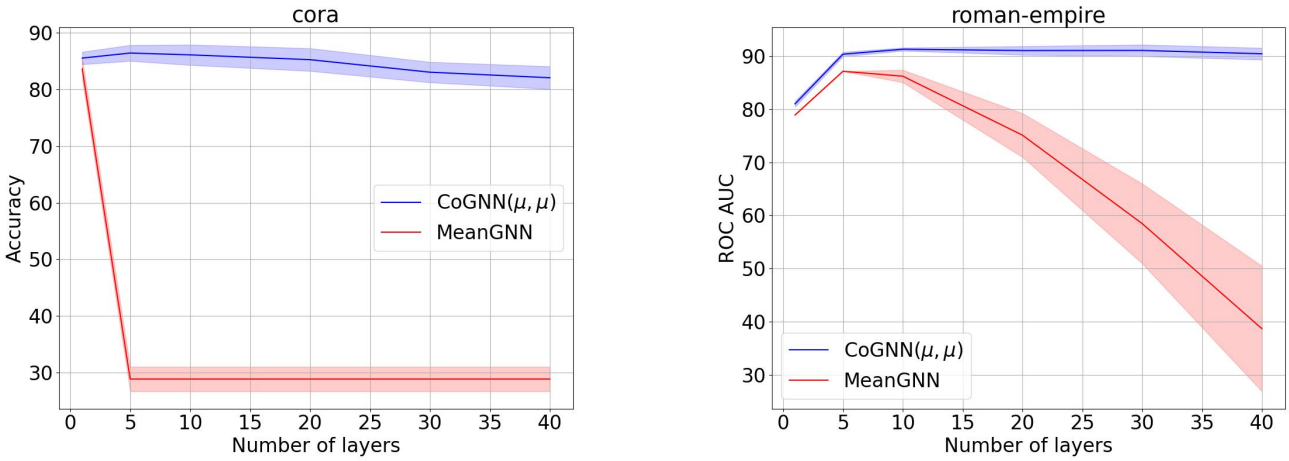


Figure 7: The accuracy of CO-GNN(μ, μ) and MEANGNN on cora (**left**) and on roman-empire (**right**) for an increasing number of layers.

C.5. Over-smoothing Experiments

Section 5.1 explains that CO-GNNs can mitigate the over-smoothing phenomenon, through the choice of BROADCAST or ISOLATE actions. To validate this, we experiment with an increasing number of layers of CO-GNN(μ, μ) and MEANGNN over the cora and roman-empire datasets.

Setup. We evaluate CO-GNN(μ, μ) and MEANGNN over the cora and roman-empire datasets, following the 10 data splits of Pei et al. (2020) and Platonov et al. (2023), respectively. We report the accuracy and standard deviation.

Results. Figure 7 indicates that the performance is generally retained for deep models and that CO-GNNs are effective in alleviating the over-smoothing phenomenon even though their base GNNs suffer from performance deterioration already with a few layers.

D. Runtime Analysis

Consider a GCN model with L layers and a hidden dimension of d on an input graph $G = (V, E, \boldsymbol{X})$. Wu et al. (2019) has shown the time complexity of this model to be $\mathcal{O}(Ld(|E|d + |V|))$. To extend this analysis to CO-GNNs, let us consider a CO-GNN$(*, *)$ architecture composed of:

- a GCN environment network η with L_η layers and hidden dimension of d_η, and
- a GCN action network π with L_π layers and hidden dimension of d_π.

A single CO-GNN layer first computes the actions for each node by feeding node representations through the action network π, which is then used in the aggregation performed by the environment layer. This means that the time complexity of a single CO-GNN layer is $\mathcal{O}(L_\pi d_\pi(|E|d_\pi + |V|) + d_\eta(|E|d_\eta + |V|))$. The time complexity of the whole CO-GNN architecture is then $\mathcal{O}(L_\eta L_\pi d_\pi(|E|d_\pi + |V|) + L_\eta d_\eta(|E|d_\eta + |V|))$.

Typically, the hidden dimensions of the environment network and action network match. In all of our experiments, the depth of the action network L_π is much smaller (typically ≤ 3) than that of the environment network L_η. Therefore, assuming $L_\pi << L_\eta$ we get that a runtime complexity of $\mathcal{O}(L_\eta d_\eta(|E|d_\eta + |V|))$, matching the runtime of a GCN model.

To empirically confirm the efficiency of CO-GNNs, we report in Figure 8 the duration of a forward pass of a CO-GNN$(*, *)$ and GCN with matching hyperparameters across multiple datasets. From Figure 8, it is evident that the increase in runtime is linearly related to its corresponding base model with R^2 values higher or equal to 0.98 across 4 datasets from different domains. Note that, for the datasets IMDB-B and PROTEINS, we report the average forward duration for a single graph in a batch.

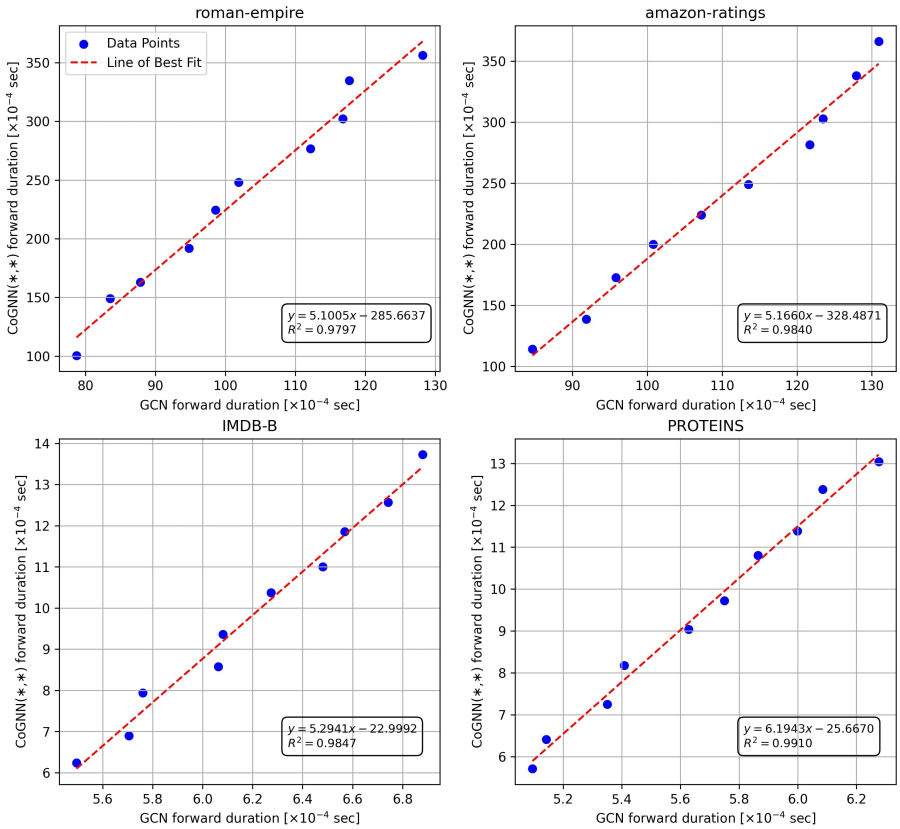


Figure 8: Empirical runtimes: CO-GNN$(*,*)$ forward pass duration as a function of GCN forward pass duration.

E. Further Details of the Experiments Reported in the Paper

E.1. The Gumbel Distribution and the Gumbel-softmax Temperature

The Gumbel distribution is used to model the distribution of the maximum (or the minimum) of a set of random variables. Its probability density function has a distinctive, skewed shape, with heavy tails, making it a valuable tool for analyzing and quantifying the likelihood of rare and extreme occurrences. By applying the Gumbel distribution to the logits or scores associated with discrete choices, the Gumbel-Softmax estimator transforms them into a probability distribution over the discrete options. The probability density function of a variable that follows $X \sim \mathrm{Gumbel}(0,1)$ is $f(x) = e^{-x+e^{-x}}$ (Figure 9).

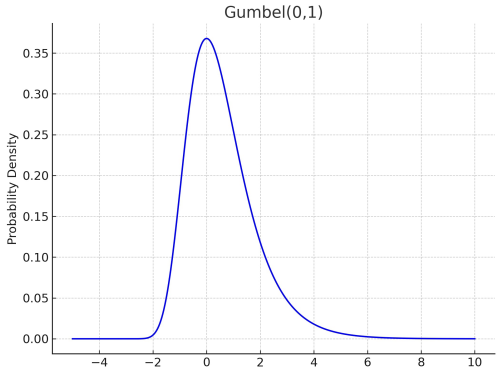


Figure 9: The pdf $f(x) = e^{-x+e^{-x}}$ of $\mathrm{Gumbel}(0,1)$.

The Straight-through Gumbel-softmax estimator is known to benefit from learning an inverse-temperature before sampling an action, which we use in our experimental setup. For a given graph $G = (V, E, \boldsymbol{X})$ the inverse-temperature of node $v \in V$ is estimated by applying a bias-free linear layer $L : \mathbb{R}^d \to \mathbb{R}$ to the intermediate representation $\boldsymbol{h} \in \mathbb{R}^d$. To ensure the temperature is positive, an approximation of the ReLU function with a bias hyperparameter $\tau \in \mathbb{R}$ is subsequently applied:

$$\frac{1}{\tau(\boldsymbol{h})} = \log\left(1 + \exp\left(\omega^T \boldsymbol{h}\right)\right) + \tau_0$$

where τ_0 controls the maximum possible temperature value.

E.2. Dataset Statistics

The statistics of the real-world long-range, node-based, and graph-based benchmarks used can be found in Tables 6 to 9.

Table 6: Statistics of the heterophilic node classification benchmarks.

	roman-empire	amazon-ratings	minesweeper	tolokers	questions
# nodes	22662	24492	10000	11758	48921
# edges	32927	93050	39402	519000	153540
# node features	300	300	7	10	301
# classes	18	5	2	2	2
edge homophily	0.05	0.38	0.68	0.59	0.84
metrics	ACC	ACC	AUC-ROC	AUC-ROC	AUC-ROC

Table 7: Statistics of the long-range graph benchmarks (LRGB).

	Peptides-func
# graphs	15535
# average nodes	150.94
# average edges	307.30
# classes	10
metrics	AP

Table 8: Statistics of the graph classification benchmarks.

	IMDB-B	IMDB-M	REDDIT-B	NCI1	PROTEINS	ENZYMES
# graphs	1000	1500	2000	4110	1113	600
# average nodes	19.77	13.00	429.63	29.87	39.06	32.63
# average edges	96.53	65.94	497.75	32.30	72.82	64.14
# classes	2	3	2	2	2	6
metrics	ACC	ACC	ACC	ACC	ACC	ACC

Table 9: Statistics of the homophilic node classification benchmarks.

	pubmed	cora
# nodes	18717	2708
# edges	44327	5278
# node features	500	1433
# classes	3	6
edge homophily	0.80	0.81
metrics	ACC	ACC

E.3. ROOTNEIGHBORS: Dataset Generation

In Section 6.1, we compare CO-GNNs to a class of MPNNs on a dedicated synthetic dataset ROOTNEIGHBORS in order to assess the model's ability to redirect the information flow. ROOTNEIGHBORS consists of 3000 trees of depth 2 with random node features of dimension $d = 5$ which is generated as follows:

- **Features**: Each feature is independently sampled from a uniform distribution $U[-2, 2]$.

- **Level-1 Nodes**: The number of nodes in the first level of each tree in the train, validation, and test set is sampled from a uniform distribution $U[3, 10]$, $U[5, 12]$, and $U[5, 12]$ respectively. Then, the degrees of the level-1 nodes are sampled as follows:
  - The number of level-1 nodes with a degree of 6 is sampled independently from a uniform distribution $U[1, 3], U[3, 5], U[3, 5]$ for the train, validation, and test set, respectively.
  - The degree of the remaining level-1 nodes are sampled from the uniform distribution $U[2, 3]$.

We use a train, validation, and test split of equal size.

E.4. Hyperparameters for all Experiments

In Tables 10 to 14, we report the hyperparameters used in our experiments.

Table 10: Hyperparameters used for ROOTNEIGHBORS and CYCLES.

	ROOTNEIGHBORS	CYCLES
η # layers	1	2
η dim	16, 32	32
π # layers	1, 2	6
π dim	8, 16	32
learned temp	✓	-
temp	-	1
τ_0	0.1	-
# epochs	10000	1000
dropout	0	0
learn rate	10^{-3}	10^{-3}
batch size	-	14
pooling	-	sum

Table 11: Hyperparameters used for the heterophilic node classification benchmarks.

	roman-empire	amazon-ratings	minesweeper	tolokers	questions
η # layers	5-12	5-10	8-15	5-10	5-9
η dim	128,256,512	128,256	32,64,128	16,32	32,64
π # layers	1-3	1-6	1-3	1-3	1-3
π dim	4,8,16	4,8,16,32	4,8,16,32,64	4,8,16,32	4,8,16,32
learned temp	✓	✓	✓	✓	✓
τ_0	0,0.1	0,0.1	0,0.1	0,0.1	0,0.1
# epochs	3000	3000	3000	3000	3000
dropout	0.2	0.2	0.2	0.2	0.2
learn rate	$3 \cdot 10^{-3}, 3 \cdot 10^{-5}$	$3 \cdot 10^{-4}, 3 \cdot 10^{-5}$	$3 \cdot 10^{-3}, 3 \cdot 10^{-5}$	$3 \cdot 10^{-3}$	$10^{-3}, 10^{-2}$
activation function	GeLU	GeLU	GeLU	GeLU	GeLU
skip connections	✓	✓	✓	✓	✓
layer normalization	✓	✓	✓	✓	✓

Table 12: Hyperparameters used for the long-range graph benchmarks (LRGB).

	Peptides-func
η # layers	5-9
η dim	200,300
π # layers	1-3
π dim	8,16,32
learned temp	✓
τ_0	0.5
# epochs	500
dropout	0
learn rate	$3 \cdot 10^{-4}, 10^{-3}$
# decoder layer	2,3
# warmup epochs	5
positional encoding	LapPE, RWSE
batch norm	✓
skip connections	✓

Table 13: Hyperparameters used for social networks and proteins datasets.

	IMDB-B	IMDB-M	REDDIT-B	REDDIT-M	NCI1	PROTEINS	ENZYMES
η # layers	1	1	3,6	6	2,5	3,5	1,2
η dim	32,64	64,256	128,256	64, 128	64,128,256	64	128,256
π # layers	2	3	1,2	1	2	1,2	1
π dim	16,32	16	16,32	16	8, 16	8	8
learned temp.	✓	✓	✓	✓	✓	✓	✓
τ_0	0.1	0.1	0.1	0.1	0.5	0.5	0.5
# epochs	5000	5000	5000	5000	3000	3000	3000
dropout	0.5	0.5	0.5	0.5	0	0	0
learn rate	10^{-4}	10^{-3}	10^{-3}	10^{-4}	10^{-3},10^{-2}	10^{-3}	10^{-3}
pooling	mean	mean	mean	mean	mean	mean	mean
batch size	32	32	32	32	32	32	32
scheduler step size	50	50	50	50	50	50	50
scheduler gamma	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Table 14: Hyperparameters used for the homophilic node classification benchmarks.

	pubmed	citeseer
η # layers	1-3	1-3
η dim	32,64,128	32,64,128
π # layers	1-3	1-3
π dim	4,8,16	4,8,16
temperature	0.01	0.01
τ_0	0.1	0.1
# epochs	2000	2000
dropout	0.5	0.5
learn rate	$5 \cdot 10^{-3}, 10^{-2}, 5 \cdot 10^{-2}$	$5 \cdot 10^{-3}, 10^{-2}, 5 \cdot 10^{-2}$
learn rate decay	$5 \cdot 10^{-6}, 5 \cdot 10^{-4}$	$5 \cdot 10^{-6}, 5 \cdot 10^{-4}$
activation function	ReLU	ReLU

F. Visualizing the Actions

We extend the discussion about Co-GNNs dynamic topology over the Minesweeper dataset in Section 7.2 and present the evolution of the graph topology from layer $\ell = 1$ to layer $\ell = 8$.

Setup. We train a 10-layered Co-GNN(μ, μ) model and present the evolution of the graph topology from layer $\ell = 1$ to layer $\ell = 8$. We choose a node (black), and at every layer ℓ, we depict its neighbors up to distance 10. In this visualization, nodes which are mines are shown in red, and other nodes in blue. The features of non-mine nodes (indicating the number of neighboring mines) are shown explicitly whereas the nodes whose features are hidden are labeled with a question mark. For each layer ℓ, we gray out the nodes whose information cannot reach the black node with the remaining layers available.

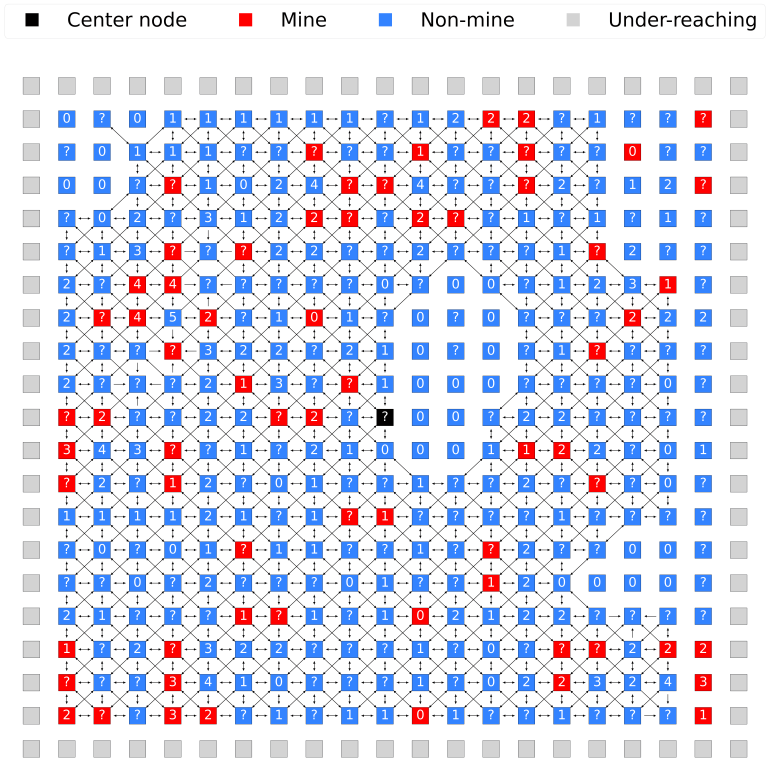


Figure 10: The 10-hop neighborhood at layer $\ell = 1$.

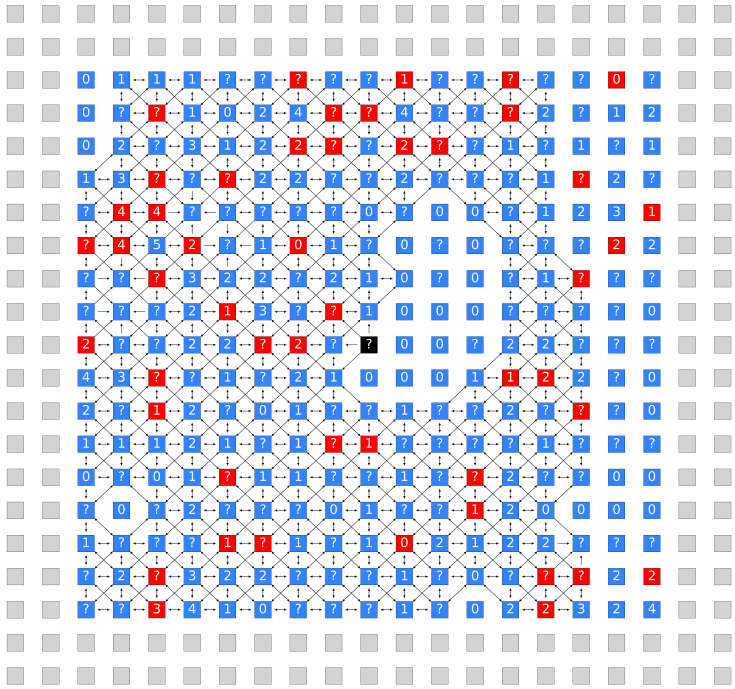


Figure 11: The 10-hop neighborhood at layer $\ell = 2$.

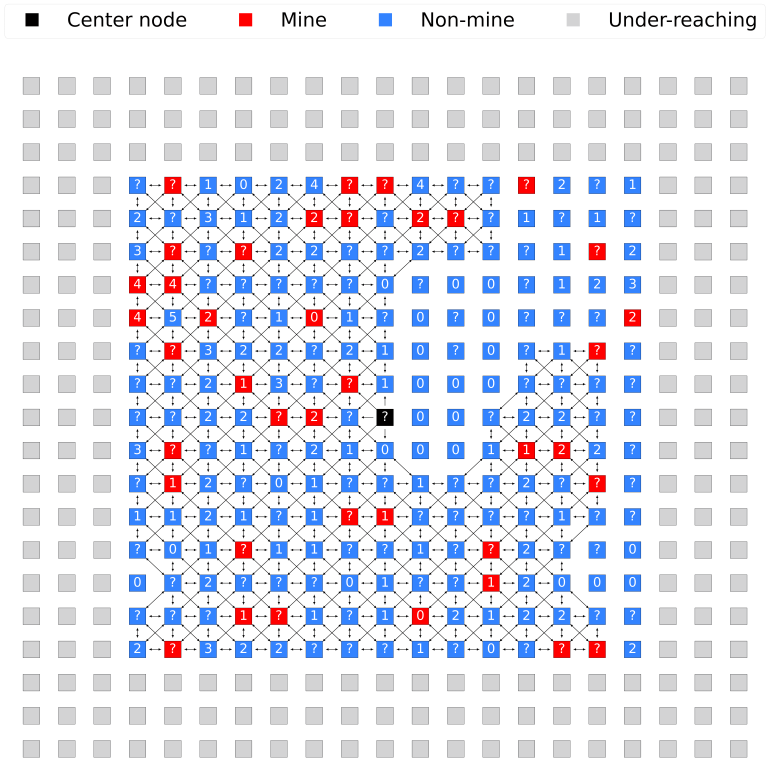


Figure 12: The 10-hop neighborhood at layer $\ell = 3$.

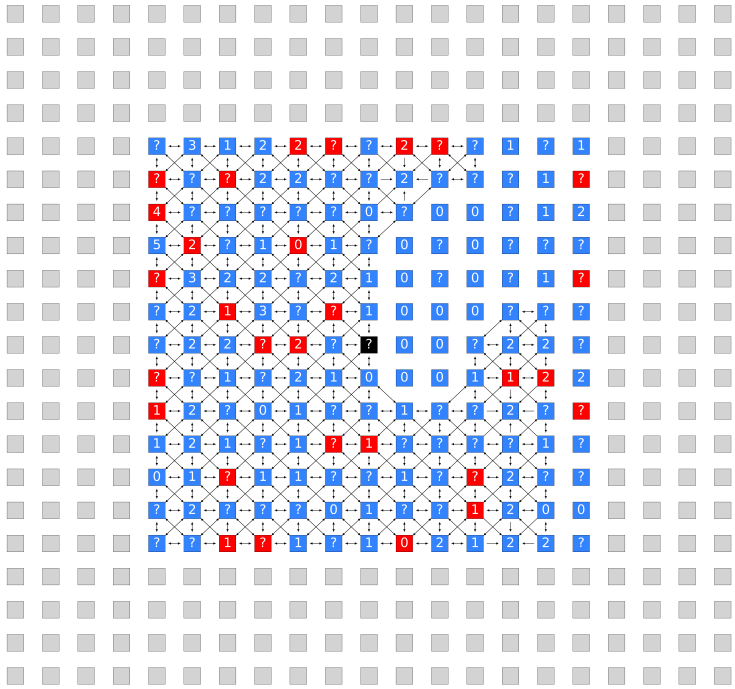


Figure 13: The 10-hop neighborhood at layer $\ell = 4$.

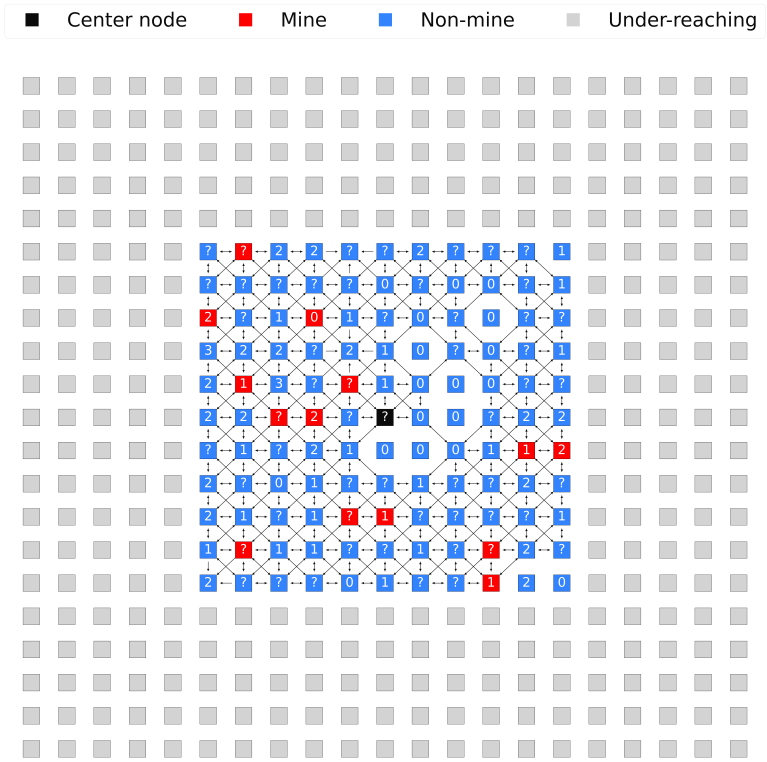


Figure 14: The 10-hop neighborhood at layer $\ell = 5$.

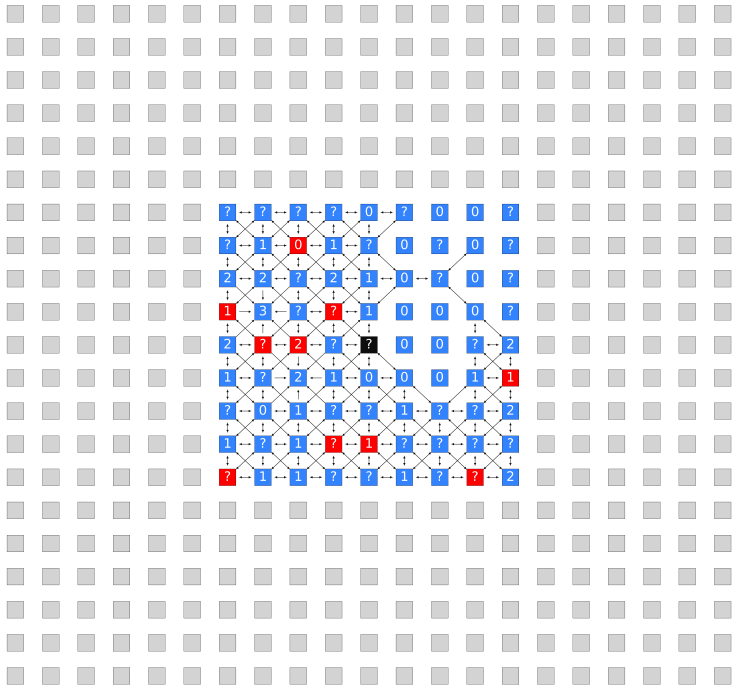


Figure 15: The 10-hop neighborhood at layer $\ell = 6$.

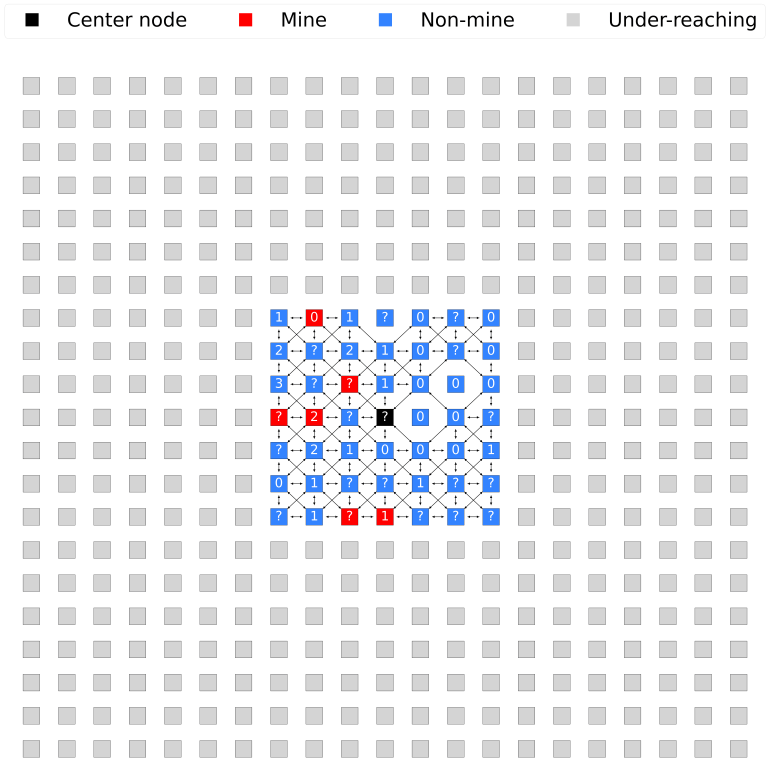


Figure 16: The 10-hop neighborhood at layer $\ell = 7$.

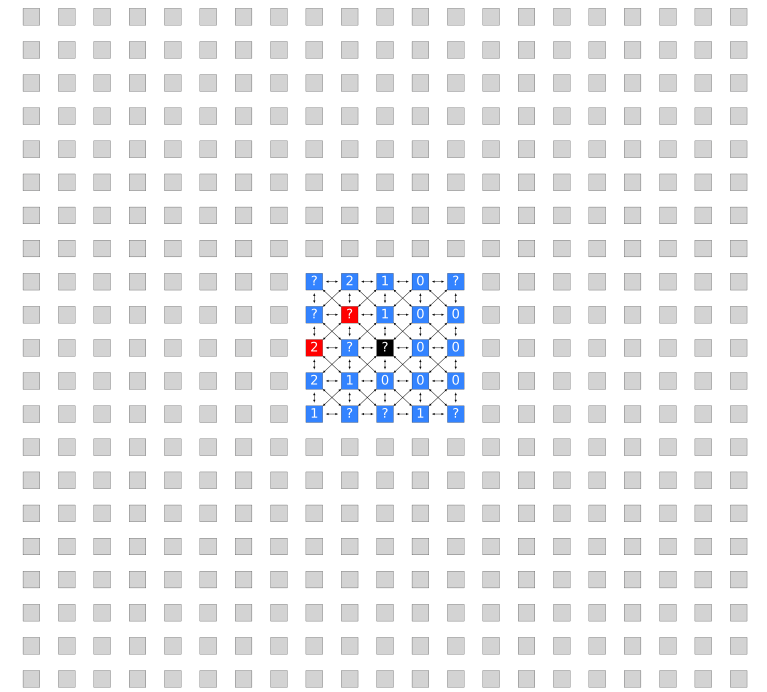


Figure 17: The 10-hop neighborhood at layer $\ell = 8$.