Supplementary Material

A. Proof of Proposition 1

Proposition 1. *[E-invariant Representation's Effect on OOD Classification] Consider a permutation-invariant graph representation* $\Gamma : \cup_{n=1}^{\infty}\{0,1\}^{n\times n} \times \mathbb{X}^n \to \mathbb{R}^d$, $d \geq 1$, *and a downstream function* $\rho : \mathbb{Y} \times \mathbb{R}^d \to [0,1]$ *(e.g., a feedforward neural network (MLP) with softmax outputs) such that, for some* $\epsilon, \delta > 0$, *the generalization error over the training distribution is:* $\forall y \in \mathbb{Y}$,

$$\mathrm{P}(\ |\mathrm{P}(Y = y|\mathcal{G}^{tr}_{N^{tr}}) - \rho(y, \Gamma(\mathcal{G}^{tr}_{N^{tr}}))|\ \leq \epsilon) \geq 1 - \delta,$$

Γ *is said to be* **environment-invariant (E-invariant)** *if* $\forall e \in supp(E^{tr}), \forall e^{\dagger} \in supp(E^{te})$,

$$\Gamma(\mathcal{G}^{tr}_{N^{tr}}|E^{tr} = e) = \Gamma(\mathcal{G}^{te}_{N^{te}}|E^{te} = e^{\dagger}).$$

If Γ *is E-invariant, then the OOD test error is the same as the generalization error over the training distribution, i.e.,* $\forall y \in \mathbb{Y}$,

$$\mathrm{P}(|\mathrm{P}(Y = y|\mathcal{G}^{te}_{N^{te}}) - \rho(y, \Gamma(\mathcal{G}^{te}_{N^{te}}))| \leq \epsilon) \geq 1 - \delta. \quad (2)$$

Proof. First note that Y is only a function of W and an independent random noise (following Definitions 1 and 2, depicted in Figure 1). Therefore, Y is E-invariant, and thus

$$\mathrm{P}(Y|\mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}} = G^{\mathrm{te}}_{N^{\mathrm{te}}}) = \mathrm{P}(Y|\mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}} = G^{\mathrm{tr}}_{N^{\mathrm{tr}}}),$$

since the observed graphs in test ($G^{\mathrm{te}}_{N^{\mathrm{te}}}$) and training ($G^{\mathrm{tr}}_{N^{\mathrm{tr}}}$) only differ due to the change of environments while sharing the same graphon variable W and the other random variables.

The definition of E-invariance states that $\forall e \in \mathrm{supp}(E^{\mathrm{tr}})$, $\forall e^{\dagger} \in \mathrm{supp}(E^{\mathrm{te}})$,

$$\Gamma(\mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}|E^{\mathrm{tr}} = e) = \Gamma(\mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}}|E^{\mathrm{te}} = e^{\dagger}).$$

So, the E-invariance of Γ yields

$$\rho(y, \Gamma(\mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}})) = \rho(y, \Gamma(\mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})),$$

concluding our proof. □

B. Proof of Theorem 1

Theorem 1 (Approximately E-invariant Graph Representation)**.** *Let* $\mathcal{G}^{tr}_{N^{tr}}$ *and* $\mathcal{G}^{te}_{N^{te}}$ *be two samples of graphs of sizes* N^{tr} *and* N^{te} *from the training and test distributions, respectively, both defined over the same graphon variable* W *and satisfying Definitions 1 and 2. Assume the vertex attribute function* $g_X(\cdot,\cdot)$ *of Definitions 1 and 2 is invariant to* E^{tr} *and* E^{te} *(the reason for this assumption will be clear later). Let* $||\cdot||_{\infty}$ *denote the L-infinity norm. For any integer* $k \leq \min(N^{tr}, N^{te})$, *and any constant* $0 < \epsilon < 1$,

$$\mathrm{P}(\|\Gamma_{\textit{1-hot}}(\mathcal{G}^{tr}_{N^{tr}}) - \Gamma_{\textit{1-hot}}(\mathcal{G}^{te}_{N^{te}})\|_{\infty} > \epsilon) \leq 2|\mathcal{F}_{\leq k}|(\exp(-\frac{\epsilon^2 N^{tr}}{8k^2}) + \exp(-\frac{\epsilon^2 N^{te}}{8k^2})). \quad (9)$$

Proof. We first replace t_{ind} by t_{inj}, which is defined by

$$t_{\mathrm{inj}}(F_k, \mathcal{G}^*_{N^*}) = \frac{\mathrm{inj}(F_k, \mathcal{G}^*_{N^*})}{N^*!/(N^* - k)!}, \quad (10)$$

where $\mathrm{inj}(F_k, \mathcal{G}^*_{N^*})$ is the number of injective homomorphisms of F_k into $\mathcal{G}^*_{N^*}$. Then, we know from Lovász & Szegedy (2006, Theorem 2.5) that for unattributed graphs $\mathcal{G}^*_{N^*}$,

$$\mathrm{P}(|t_{\mathrm{inj}}(F_k, \mathcal{G}^*_{N^*}) - t(F_k, W)| > \epsilon) \leq 2\exp(-\frac{\epsilon^2}{2k^2}N^*), \quad (11)$$

where W is the graphon function as illustrated in Definitions 1 and 2. As defined in Lovász & Szegedy (2006),

$$t(F_k, W) = \int_{[0,1]^k} \prod_{ij \in E(F_k)} W(x_i, x_j) dx_1 \cdots dx_k,$$

where $E(F_k)$ denotes the edge set of F_k. This bound shows that $t_{\mathrm{inj}}(F_k, \mathcal{G}^*_{N^*})$ converges to $t(F_k, W)$ as $N^* \to \infty$. Actually, we can get similar bounds for t_{ind} using a similar proof technique. Although the value it converges to is different, the difference between values is preserved as it will be proved in the following text. More importantly, it can be extended to vertex-attributed graphs under our SCM assumptions depicted in Definitions 1 and 2.

We can have this extension because, for the vertex-attributed graphs in Definitions 1 and 2, g_X operates on attributed graphs similarly as the graphon does on unattributed graphs. We can consider the graph generation procedure as first generating the underlying structure, and then adding vertex attribute accordingly to its corresponding random graphon value $U_v \in \mathrm{Uniform}(0,1)$ and graphon W. $g_X(\cdot,\cdot)$ being invariant to E^{tr} and E^{te} means for any two environments $e \in \mathrm{supp}(E^{\mathrm{tr}}), e^{\dagger} \in \mathrm{supp}(E^{\mathrm{te}})$, $g_X(e,\cdot) = g_X(e^{\dagger},\cdot)$.

Then for a given vertex-attributed graph F_k with k vertices, and a given (whole) graph size N^*, we can define ϕ as an

induced map $\phi : [k] \to [N^*]$, which can be thought about as how the k vertices in F_k are mapped to the vertices in $\mathcal{G}^*_{N^*}$. Define $C_\phi = 1$ if ϕ is a homomorphism from F_k to the W-random graph $\mathcal{G}^*_{N^*}$, otherwise $C_\phi = 0$. We define $\mathcal{G}^*_m$ as the subgraph of $\mathcal{G}^*_{N^*}$ induced by vertices $\{1, ..., m\}$. Note here m has two meanings. First, it represents the m-th vertex. Second, it indicates the size of the subgraph. We define $B_m = \frac{1}{\binom{N^*}{k}} \sum_\phi \mathrm{P}(C_\phi = 1 | \mathcal{G}^*_m), 0 \le m \le N^*$ as the expected induced homomorphism densities once we observe the subgraph $\mathcal{G}^*_m$. Here $B_0 = \frac{1}{\binom{N^*}{k}} \sum_\phi \mathrm{P}(C_\phi = 1)$ denotes the expectation before we observe any vertices.

B_m is a martingale for unattributed graphs (Lovász & Szegedy, 2006, Theorem 2.5). And since in Definitions 1 and 2 we also use the graphon W and g_X operates on attributed graphs using the graphon W and U_v, it is also a martingale for vertex-attributed graphs based on our definitions. We do not need to care about the environment variable E^* here because the function g_X is invariant to E^* and, therefore, it can be treated as a constant. Then,

$$
\begin{aligned}
&|B_m - B_{m-1}| \\
=&\frac{1}{\binom{N^*}{k}} |\sum_\phi \mathrm{P}(C_\phi = 1 | \mathcal{G}^*_m) - \mathrm{P}(C_\phi = 1 | \mathcal{G}^*_{m-1})| \\
\le&\frac{1}{\binom{N^*}{k}} \sum_\phi |\mathrm{P}(C_\phi = 1 | \mathcal{G}^*_m) - \mathrm{P}(C_\phi = 1 | \mathcal{G}^*_{m-1})|.
\end{aligned}
$$

Here, for each $\phi : [k] \to [N^*]$ that does not contain the value m in its image (which means no vertex in F_k is mapped to the m-th vertex in $\mathcal{G}^*_{N^*}$), the difference is 0. For all other terms, the terms are at most 1. Thus,

$$
|B_m - B_{m-1}| \le \frac{\binom{N^*-1}{k-1}}{\binom{N^*}{k}} = \frac{k}{n}.
$$

By definition, $B_0 = \frac{1}{\binom{N^*}{k}} \sum_\phi \mathrm{P}(C_\phi = 1) = t^*(F_k, W)$, and $B_{N^*} = \frac{1}{\binom{N^*}{k}} \mathrm{ind}(F_k, \mathcal{G}^*_{N^*}) = t_{\mathrm{ind}}(F_k, \mathcal{G}^*_{N^*})$, where $t^*(F_k, W)$ is defined as B_0, is the expected induced homomorphism densities if we only know the graphon W and we did not observe any vertex in the graph.

Then, we can use Azuma's inequality for Martingales,

$$
\begin{aligned}
\mathrm{P}(B_{N^*} - B_0 > \epsilon) &\le \exp(-\frac{\epsilon^2}{2N^*(k/N^*)^2}) \\
&= \exp(-\frac{\epsilon^2}{2k^2} N^*).
\end{aligned}
$$

Since $B_{N^*} = t_{\mathrm{ind}}(F_k, \mathcal{G}^*_{N^*})$, and $B_0 = t^*(F_k, W)$, we get the similar bound as in Equation (11),

$$
\mathrm{P}(|t_{\mathrm{ind}}(F_k, \mathcal{G}^*_{N^*}) - t^*(F_k, W)| > \epsilon) \le 2 \exp(-\frac{\epsilon^2}{2k^2} N^*).
$$

Since $|t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - t^*(F_k, W)| \le \frac{\epsilon}{2}$, $|t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}}) - t^*(F_k, W)| \le \frac{\epsilon}{2}$ imply $|t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})| \le \epsilon$, we have,

$$
\begin{aligned}
&\mathrm{P}(|t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})| > \epsilon) \\
&= 1 - \mathrm{P}(|t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})| \le \epsilon) \\
&\le 1 - \mathrm{P}(|t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - t^*(F_k, W)| \le \frac{\epsilon}{2}) \\
&\qquad \cdot \mathrm{P}(|t_{\mathrm{ind}}(F_k, \mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}}) - t^*(F_k, W)| \le \frac{\epsilon}{2}) \\
&\le 1 - (1 - 2\exp(-\frac{\epsilon^2}{8k^2} N^{\mathrm{tr}}))(1 - 2\exp(-\frac{\epsilon^2}{8k^2} N^{\mathrm{te}})) \\
&= 2(\exp(-\frac{\epsilon^2}{8k^2} N^{\mathrm{tr}}) + \exp(-\frac{\epsilon^2}{8k^2} N^{\mathrm{te}})) \\
&\qquad - 4\exp(-\frac{\epsilon^2}{8k^2}(N^{\mathrm{tr}} + N^{\mathrm{te}})) \\
&\le 2(\exp(-\frac{\epsilon^2}{8k^2} N^{\mathrm{tr}}) + \exp(-\frac{\epsilon^2}{8k^2} N^{\mathrm{te}})).
\end{aligned} \tag{12}
$$

Then we know,

$$
\begin{aligned}
&\mathrm{P}(||\Gamma_{\text{1-hot}}(\mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - \Gamma_{\text{1-hot}}(\mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})||_\infty \le \epsilon) \\
&= \mathrm{P}(|t_{\mathrm{ind}}(F_{k'}, \mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - t_{\mathrm{ind}}(F_{k'}, \mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})| \le \epsilon, \forall F_{k'} \in \mathcal{F}_{\le k}) \\
&\ge 1 - \sum_{F_{k'} \in \mathcal{F}_{\le k}} \mathrm{P}(|t_{\mathrm{ind}}(F_{k'}, \mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - t_{\mathrm{ind}}(F_{k'}, \mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})| > \epsilon) \\
&\ge 1 - 2|\mathcal{F}_{\le k}|(\exp(-\frac{\epsilon^2 N^{\mathrm{tr}}}{8k^2}) + \exp(-\frac{\epsilon^2 N^{\mathrm{te}}}{8k^2})).
\end{aligned} \tag{13}
$$

It follows from the Bonferroni inequality that $\mathrm{P}(\cap_{i=1}^N A_i) \ge 1 - \sum_{i=1}^N \mathrm{P}(\tilde{\mathrm{A}}_\mathrm{i})$, where A_i and its complement $\tilde{A}_i$ are any events. Therefore,

$$
\begin{aligned}
&\mathrm{P}(||\Gamma_{\text{1-hot}}(\mathcal{G}^{\mathrm{tr}}_{N^{\mathrm{tr}}}) - \Gamma_{\text{1-hot}}(\mathcal{G}^{\mathrm{te}}_{N^{\mathrm{te}}})||_\infty > \epsilon) \\
&\qquad \le 2|\mathcal{F}_{\le k}|(\exp(-\frac{\epsilon^2 N^{\mathrm{tr}}}{8k^2}) + \exp(-\frac{\epsilon^2 N^{\mathrm{te}}}{8k^2})),
\end{aligned}
$$

concluding the proof. □

C. Biases in estimating induced homomorphism densities

Induced (connected) homomorphism densities of a given graph $F_{k'}$ over all possible k'-vertex ($k' \le k$) connected graphs for an N^*-vertex graph $\mathcal{G}^*_{N^*}$ are defined as

$$
\omega(F_{k'}, \mathcal{G}^*_{N^*}) = \frac{\mathrm{ind}(F_{k'}, \mathcal{G}^*_{N^*})}{\sum_{F_{k'} \in \mathcal{F}_{\le k}} \mathrm{ind}(F_{k'}, \mathcal{G}^*_{N^*})}.
$$

This is a slightly different definition from the induced homomorphism densities in Equation (3). In the main text, the denominator is the total number of possible mappings (which can include mappings that are disconnected). Here we consider the total numbers of induced mappings that are connected as is common practice.

Achieving unbiased estimates for induced (connected) homomorphism densities usually requires sophisticated methods and significant amount of time. We show that a biased estimator can also work for the GNN^+ in Equation (7) if the bias is multiplicative and the READOUT_Γ is simply the sum of the vertex embeddings. We formalize it as follows.

Proposition 2. *Assume $\hat{\omega}(F_{k'}, \mathcal{G}^*_{N^*})$ is a biased estimator for $\omega(F_{k'}, \mathcal{G}^*_{N^*})$ for any k' and k'-sized connected graphs $F_{k'}$ in a N^*-vertex $\mathcal{G}^*_{N^*}$, such that $\mathbb{E}(\hat{\omega}(F_{k'}, \mathcal{G}^*_{N^*})) = \beta(F_{k'})\omega(F_{k'}, \mathcal{G}^*_{N^*})$, where $\beta(F_{k'})$ ($\beta(\cdot) > 0$) is the bias related to the graph $F_{k'}$, and the expectation is over the sampling procedure. The expected learned representation $\mathbb{E}(\sum_{F_{k'} \in \mathcal{F}_{\leq k}} \hat{\omega}(F_{k'}, \mathcal{G}^*_{N^*}) \mathbf{1}^{\mathrm{T}}(GNN^+(F_{k'})))$ can be the same as using the true induced (connected) homomorphism densities $\omega(F_{k'}, \mathcal{G}^*_{N^*}), \forall F_{k'} \in \mathcal{F}_{\leq k}$.*

Proof. W.L.O.G, assume $\text{GNN}^+_0(F_{k'})$ is the representation we can learn from the true induced (connected) homomorphism densities $\omega(F_{k'}, \mathcal{G}^*_{N^*}), \forall F_{k'} \in \mathcal{F}_{\leq k}$. When only using the biased estimators, if we are able to learn the representation $\text{GNN}^+(F_{k'}) = \text{GNN}^+_0(F_{k'})/\beta(F_{k'})$ for all $F_{k'} \in \mathcal{F}_{\leq k}$, then we can still get the graph representation in Equation (7) the same as using the true induced (connected) homomorphism densities. This is possible because GNN^+ is proven to be a most expressive k'-vertex graph representation, thus it is able to learn any function on the graph $F_{k'}$. Then,

$$\mathbb{E}\left[\sum_{F_{k'} \in \mathcal{F}_{\leq k}} \hat{\omega}(F_{k'}, \mathcal{G}^*_{N^*}) \mathbf{1}^{\mathrm{T}}(\text{GNN}^+(F_{k'}))\right] = \sum_{F_{k'} \in \mathcal{F}_{\leq k}} \omega(F_{k'}, \mathcal{G}^*_{N^*}) \mathbf{1}^{\mathrm{T}}(\text{GNN}^+_0(F_{k'})), \quad (14)$$

where $\mathbf{1}^{\mathrm{T}}(\text{GNN}^+(F_{k'}))$ is the sum of the vertex embeddings given by the GNN^+ if it is an equivariant representation of the graph. □

D. Review of Graph Neural Networks

Graph Neural Networks (GNNs) constitute a popular class of methods for learning representations of vertices in a graph or graph-wide representations (Kipf & Welling, 2017; Atwood & Towsley, 2016; Hamilton et al., 2017; Gilmer et al., 2017; Veličković et al., 2018; Xu et al., 2019; Morris et al., 2019; You et al., 2019; Liu et al., 2019; Chami et al., 2019). Graph-wide representations can also be obtained by applying GNNs to the connected induced subgraphs in a larger graph and then averaging the resulting subgraph representations. That is, in our work, we have applied GNNs to connected induced subgraphs in a graph, and then aggregated (averaged) them to obtain the representation of the graph. We briefly summarize the idea, but more details can be found in texts such as by Hamilton (2020) and reviews by Wu et al. (2020) and Zhang et al. (2020) and the references therein.

Suppose we have a graph G with vertex set $V = \{1, \ldots, N\}$, and each vertex in our data may carry some vertex attribute (also called a *feature*). For instance, in a molecule, vertices may represent atoms, edges may represent bonds, and features may indicate the atomic number (Duvenaud et al., 2015). These vertex features can be stored in an $N \times d$ matrix $\boldsymbol{X}$, where d is the dimension of the vertex feature vector. In particular, row $v \in V$ of X_v holds the attribute associated with vertex v.

Roughly speaking, GNNs proceed by passing messages among vertices, later passing the result through a learnable function such as an MLP, and repeating $T \in \mathbb{Z}_{\geq 1}$ times. At each iteration $t = \{1, 2, \ldots, T\}$, all vertices $v \in V$ are associated with a learned vector $\boldsymbol{h}^{(t)}$. Specifically, we begin by initializing a vector as $\boldsymbol{h}^{(0)}_v = X_v$ for every vertex $v \in V$. Then, we recursively compute an update such as the following

$$\boldsymbol{h}^{(t)}_v = \text{MLP}^{(t)}\Big(\boldsymbol{h}^{(t-1)}_v, \sum_{u \in \mathcal{N}(v)} \boldsymbol{h}^{(t-1)}_u\Big), \quad \forall v \in V, \quad (15)$$

where $\mathcal{N}(v) \subseteq V$ denotes the neighborhood set of v in the graph, $\text{MLP}^{(t)}$ denotes a multi-layer perceptron, and whose superscript t indicates that the MLP at each recursion layer may have different learnable parameters. We can replace the summation with any permutation-invariant function of the neighborhood. We see that GNNs recursively update vertex states with states from their neighbors and their state from the previous recursion layer. Additionally, we can sample from the neighborhood set rather than aggregating over every neighbor. Generally speaking there is much research into the variations of this recursion step and we refer the reader to aforementioned references for details.

To learn a graph representation, we can aggregate the vertex representations using a so-called *READOUT* function defined to be permutation-invariant over the labels. A graph representation $\boldsymbol{h}_G$ by a GNN is then

$$\boldsymbol{h}_G = \text{READOUT}\Big(\{\boldsymbol{h}^{(t)}_v\}_{v,t \in V \times \{1\ldots,T\}}\Big),$$

where the vertex features $\boldsymbol{h}^{(t)}_v$ are as in Equation (15). READOUT may or may not contain learnable weights. We denote it as *XU-READOUT* to not confuse with our notation READOUT_Γ.

The entire function is differentiable and can be learned end-to-end. These models are thus typically trained with variants of Stochastic Gradient Descent. In our work, we apply this scheme over connected induced subgraphs in the graph, making them a differentiable module in our end-to-end representation scheme.

E. Further Related Work

This section provides a more in-depth discussion placing our work in the context of existing literature. We explain why existing state-of-the-art graph learning methods will struggle to extrapolate, subgraph methods, and more in Graph Neural Networks literature.

Extrapolation. Geometrically, extrapolation can be thought as reasoning beyond a convex hull of a set of training points (Hastie et al., 2012; Haffner, 2002; King & Zeng, 2006; Xu et al., 2021). However, for neural networks—and their arbitrary representation mappings—this geometric interpretation is insufficient to describe a truly broad range of tasks. Rather, extrapolations are better described through counterfactual reasoning (Neyman, 1923; Rubin, 1974; Pearl, 2009; Schölkopf, 2019).

As shown in Geirhos et al. (2020), the ability of deep neural networks to capture shortcuts for predictions tends to results in poor extrapolation performances. Therefore, specific methods or strategies must be adopted to obtain extrapolation abilities.

There are other approaches for conferring models with extrapolation abilities. These ideas have started to permeate graph literature, which we touch on here, but remain outside the scope of our systematic counterfactual modeling framework.

Incorporating domain knowledge is an intuitive approach to learn a function that predicts adequately outside of the training distribution, data collection environment, and heuristic curation. This has been used, for example, in time series forecasting (Scott Armstrong & Collopy, 1993; Armstrong et al., 2005). This can come in the form of re-expressing phenomena in a way that can be adequately and accurately represented by machine learning methods (Lample & Charton, 2020) or specifically augmenting existing general-purpose methods to task (Klicpera et al., 2020). In the context of graphs, it has been used to pre-process the graph input to make a learned graph neural network model a less complex function and thus extend beyond training data (Xu et al., 2021), although this does not necessarily fall into the framework we consider here.

Another way of moving beyond the training data is *robustness*. Relevant for deep learning systems are adversarial attacks (Papernot et al., 2017). Neural networks can be highly successful classifiers on the training data but become wildly inaccurate with small perturbations of those training examples (Goodfellow et al., 2015). This is important, say, in self-driving cars (Sitawarin et al., 2018), which can become confused by graffiti. This becomes particularly problematic when we deploy systems to real-world environments outside the training data. Learning to defend against adversarial attacks is in a way related to performing well outside the environment and curation heuristics encountered in training. An interesting possibility for future work is to explore the relationships between the two approaches.

Overfitting will compromise even in-distribution generalization. Regularization schemes such as explicit penalties are a well known and broadly applicable strategy (Hastie et al., 2012). Another implicit approach is data augmentation (Hernández-García & König, 2018), and the recent GraphCrop method proposes a scheme for graphs that randomly extracts subgraphs from certain graphs in a minibatch during training (Wang et al., 2020b). These directions differ from our own in that we seek a formulation for extrapolation even when overfitting is not necessarily a problem. Still these two approaches are both useful in the toolbox of representation learning.

We would like to point out that representation learning on *dynamic graphs* (Kazemi et al., 2020), including tasks like link prediction on growing graphs (Anonymous, 2021), is a mostly separate research direction from what we consider here (although it is now understood that temporal and static graph representations are equivalent for observational predictions (Gao & Ribeiro, 2021)). In these scenarios, there is a direct expectation that the process we model will change and evolve. For instance, knowledge bases – a form of graph encoding facts and relationships – are inevitably incomplete (Sun et al., 2018). Simply put, developments in information and society move faster than they can be curated. Another important example is recommendation systems (Kumar et al., 2019) based on evolving user-item networks. These concepts are related to the counterfactuals on graphs (Eckles et al., 2016) that we discuss. This is fundamentally different from our work where we do graph-wide learning and representation of a dataset of many graphs rather than one constantly evolving graph.

Subraph methods and Graphlet Counting Kernels. A foundational principle of our work is that, by exploiting subgraphs, we confer graph classifications models with both the ability to fit the training data and to extrapolate to graphs from a different distribution (OOD generalization). As detailed in Section 3.2, this insight follows from the Aldous-Hoover representation of jointly exchangeable distributions (graphs) (Hoover, 1979; Aldous, 1981; Kallenberg, 2006; Orbanz & Roy, 2014) and work on graph limits (Lovász, 2012). We now discuss the larger literature that uses subgraphs in machine learning.

Counting kernels (Shervashidze et al., 2009) measures the similarity between two graphs by the dot product of their normalized counts of connected induced subgraphs (graphlet). This can be used for classification via kernelized methods like Support Vector Machines (SVM).

Yanardag & Vishwanathan (2015) argues that the dot product does not capture dependence between subgraphs and extend to a general bilinear form over a learned similarity matrix. These approaches are related to the Reconstruction Conjecture, which posits graphs can be determined through knowledge of their subgraphs (Kelly et al., 1957; Ulam, 1960; Hemminger, 1969; McKay, 1997). It is known that computing a maximally expressive graph kernel, or one that is injective over the class of graphs, is as hard as the Graph Isomorphism problem, and thus intractable in general (Gärtner et al., 2003; Kriege et al., 2020). Kriege et al. (2018) shows graph properties that subgraph counting kernels fail to predict. The work then proposes a method to make them more expressive, but only for graphs without vertex attributes.

Most applications of graphlet counting do not exploit vertex attributes, and even those that do (e.g. Wale et al. (2008)) are likely to fail under a distribution shift over attributes; this is because counting each type of attributed subgraph (e.g. red clique, blue clique) is sensitive to distribution shift. In comparison, our use of GNNs confers our framework with the ability learn a compressed representation of different attributed subgraphs, tailored for the task, and extrapolate even under attribute shift. We demonstrate this in Table 2. Last, a recent work (Ye et al., 2020) proposes to pass the attributed subgraph counts to a downstream neural network model to better compress and represent the high dimensional feature space. However, with attribute shifts, it may be that the downstream layers did not see enough attributed subgraph of certain types in training to learn how to correctly represent them. We feel that it is better to *compress* the attributed signal *in the process of* representing the graph to handle these vertex features, the approach we take in this work.

There are many graph kernel methods that do not leverage subgraph counts but other features to measure graph similarity, such as the count of matching walks, e.g. Kashima et al. (2003); Borgwardt et al. (2005); Borgwardt & Kriegel (2005). The WL Kernel uses the WL algorithm to compare graphs (Shervashidze et al., 2011) and will inherit the limitations of WL GNNs like inability to represent cycles. Rieck et al. (2019) propose a persistent WL kernel that uses ideas from Topological Data Analysis (Munch, 2017) to better capture such structures when comparing graphs. Methods that do not count subgraphs will not inherit properties regarding a graph-size environment change – from our analysis of asymptotic graph theory – but all extrapolation tasks require an assumption and our framework can be applied to studying the ability of various kernel methods to extrapolate under different scenarios. Those relying on attributes to build similarities are also likely to suffer from attribute shift.

Subgraphs are studied to understand underlying mechanisms of graphs like gene regulatory networks, food webs, and the vulnerability of networks to attack, and sometimes used prognostically. A popular example investigates *motifs*, subgraphs that appear more frequently than under chance (Stone & Roberts, 1992; Shen-Orr et al., 2002; Milo et al., 2002; Mangan & Alon, 2003; Sporns & Kötter, 2004; Bascompte & Melián, 2005; Alon, 2007; Chen et al., 2013; Benson et al., 2016; Stone et al., 2019; Dey et al., 2019; Wang et al., 2020a). Although the study of motifs is along a different direction and often focus on one-graph datasets, our framework learns rich latent representations of subgraphs. Another line of work uses subgraph counts as graph similarity measures, an example being matching real-world graphs to their most similar random graph generation models (Pržulj, 2007).

Other machine learning methods based on subgraphs have also been proposed. Methods like mGCMN (Li et al., 2020), HONE (Rossi et al., 2018), and MCN (Lee et al., 2018) learn representations for vertices by extending classical methods over edges to a new neighborhood structure based on subgraphs; for instance, mGCMN runs a GNN on the new graph. These methods do not exploit all subgraphs of size k and will not learn subgraph representations in a manner consistent with our extrapolation framework. Teru et al. (2020) uses subgraphs around vertices to predict missing facts in a knowledge base. Further examples include the Subgraph Prediction Neural network (Meng et al., 2018) that predicts subgraph classes in one dynamic heterogeneous graph; counting the appearance of edges in each type of subgraph for link prediction tasks (Abuoda et al., 2019); and SEAL (Zhang & Chen, 2018) runs a GNN over subgraphs extracted around candidate edges to predict whether an edge exists. While these methods exploit small subgraphs for their effective balance between rich graph information and computational tractability, they are along an orthogonal research direction.

Graph Neural Networks. Among the many approaches for graph representation learning and classification, which include methods for vertex embeddings that are subsequently read-out into graph representations (Belkin & Niyogi, 2002; Perozzi et al., 2014; Niepert et al., 2016; Ou et al., 2016; Kipf & Welling, 2016; Grover & Leskovec, 2016; Yu et al., 2018; Qiu et al., 2018; Maron et al., 2019b;a; Wu et al., 2020; Hamilton, 2020; Chami et al., 2020), we focus our discussion and modeling on Graph Neural Network (GNN) methods (Kipf & Welling, 2017; Atwood & Towsley, 2016; Hamilton et al., 2017; Gilmer et al., 2017; Veličković et al., 2018; Xu et al., 2019; Morris et al., 2019; You et al., 2019; Liu et al., 2019; Chami et al., 2019). GNNs are trained end-to-end, can straightforwardly provide latent graph representations for graphs of any size, easily handle

vertex/edge attributes, are computationally efficient, and constitute a state-of-the-art method. However, GNNs lack extrapolation capabilities due also to their inability to learn latent representations that capture the topological structure of the graph (Xu et al., 2019; Morris et al., 2019; Garg et al., 2020; Sato, 2020). Relevantly, many cannot count the number of subgraphs such as triangles (3-cliques) in a graph (Arvind et al., 2020; Chen et al., 2020). In general, our theory of extrapolating in graph tasks requires properly capturing graph structure. In our work we consider GIN (Xu et al., 2019), GCN (Kipf & Welling, 2017) and PNA (Corso et al., 2020) as baseline GNN models. GIN and GCN are some of the most widely used models in literature. PNA generalizes different GNN models by considering multiple neighborhood aggregation schemes. Note that since we compare against PNA we do not need to consider other neighboorhod aggregation schemes in GNNs, as studied in Veličković et al. (2020). To test whether more expressive models are able to extrapolate, we employ RPGIN (Murphy et al., 2019). In our experiments, we show that these state-of-the-art methods are expressive in-distribution but fail to extrapolate.

F. Experiments

In this appendix we present the details of the experimental section, discussing the hyperparameters that have been tuned. Training was performed on NVIDIA GeForce RTX 2080 Ti, GeForce GTX 1080 Ti, TITAN V, and TITAN Xp GPUs.

F.1. Model implementation

All neural network approaches, including the models proposed in this paper, are implemented in PyTorch (Paszke et al., 2019) and Pytorch Geometric (Fey & Lenssen, 2019).

Our GIN (Xu et al., 2019), GCN (Kipf & Welling, 2017) and PNA (Corso et al., 2020) implementations are based on their Pytorch Geometric implementations. We consider sum, mean, and max READOUTs as proposed by Xu et al. (2021) for extrapolations (denoted by *XU-READOUT*). For RPGIN (Murphy et al., 2019), we implement the permutation and concatenation with one-hot identifiers (of dimension 10) and use GIN as before. Other than a few hyperparameters and architectural choices, we use standard choices (e.g. Hu et al. (2020)) for neural network architectures. If the graphs are unattributed, we follow convention and assign a constant $\mathbf{1}$ dummy feature to every vertex.

We use the WL graph kernel implementations provided by the *graphkernels* package (Sugiyama et al., 2017). All kernel methods use a Support Vector Machine on scikit-learn (Pedregosa et al., 2011).

The Graphlet Counting kernel (GC kernel), as well as our own procedure, relies on being able to efficiently count attributed or unattributed connected induced homomorphisms within the graph. We use ESCAPE (Pinar et al., 2017) and R-GPM (Teixeira et al., 2018) as described in the main text. The source code of ESCAPE is available online and the authors of Teixeira et al. (2018) provided us their code. We pre-process each graph beforehand and save the obtained estimated induced homomorphism densities. Note that R-GPM takes around 20 minutes per graph in the worst case considered, but graphs can be pre-processed in parallel. ESCAPE takes up to one minute per graph.

All the models learn graph representations $\Gamma(\mathcal{G}^*_{N^*})$, which we pass to a L-hidden layer feedforward neural network (MLP) with softmax outputs ($L \in \{0, 1\}$ depending on the task) to obtain the prediction. For Γ_{GIN}, and Γ_{RPGIN}, we use respectively GIN and RPGIN as our base models to obtain latent representations for each k-sized connected induced subgraph. Then, we sum over the latent representations, each weighted by its corresponding induced homomorphism density, to obtain the graph representation. For $\Gamma_{\text{1-hot}}$, the representation $\Gamma_{\text{1-hot}}(\mathcal{G}^*_{N^*})$ is a vector containing densities of each (possibly attributed) k-sized connected subgraph. To map this into a graph representation, we apply $\Gamma_{\text{1-hot}}(\mathcal{G}^*_{N^*})^{\mathrm{T}}\boldsymbol{W}$ where $\boldsymbol{W}$ is a learnable weight matrix whose rows are subgraph representations. Note that this effectively learns a unique weight vector for each subgraph type.

We use the Adam optimizer to optimize all the neural network models. When an in-distribution validation set is available (see below), we use the weights that achieve best validation-set performance for prediction. Otherwise, we train for a fixed number of epochs.

The specifics of hyperparameter grids and downstream architectures are discussed in each section below.

F.2. Schizophrenia Task: Size extrapolation

The results of these experiments are reported in Table 1 (left). The data was graciously provided by the authors of De Domenico et al. (2016), which they pre-processed from publicly available data from The Center for Biomedical Research Excellence. There are 145 graphs which represent the functional connectivity brain networks of 71 schizophrenic patients and 74 healthy controls. Each graph has 264 vertices representing spherical regions of interest (ROIs). Edges represent functional connectivity. Originally, edges reflected a time-series coherence between regions. If the coherence between signals from two regions was above a certain threshold, the authors created a weighted edge. Otherwise, there is no edge. For simplicity, we converted these to unweighted edges. Extensive pre-processing must be done over fMRI data to create brain graphs. This includes discarding signals from certain ROIs. As described

by the authors, these choices make highly significant impacts on the resulting graph. We refer the reader to the paper (De Domenico et al., 2016). Note that there are numerous methods for constructing a brain graph, and in ways that change the number of vertices. The measurement strategy taken by the lab can result in measuring about 500 ROIs, 1000 ROIs, or 264 as in the case we consider (Hagmann et al., 2007; Wedeen et al., 2005; De Domenico et al., 2016).

For our purposes, we wish to create an extrapolation task, where a change in environment leads to an extrapolation set that contains smaller graphs. For this, we randomly select 20 of the 145 graphs in the dataset, balanced among the healthy and schizophrenic patients, to be used as test. For each healthy-group graph in these 20 graphs, we sample (with replacement) $\lfloor 0.4 \times 264 \rfloor$ vertices to be removed. In average, the new size for the healthy-group graphs in these 20 graphs is 178.2.

We hold out the test graphs that are later used to assess the extrapolation capabilities. Over the remaining data, we use a stratified 5-fold cross-validation to choose the hyperparameters and to report the validation accuracy.

Once the best hyperparameters are chosen, we re-train the model on the entire training data using 10 different initialization seeds, and predict on the test.

For Γ_{GIN} and Γ_{RPGIN}, in their GNNs, the aggregation MLP of Equation (15) has hidden neurons chosen among $\{32, 64, 128, 256\}$ and number of layers (i.e. recursions of message-passing) among $\{1, 2\}$. The learning rate is chosen in $\{0.001, 0.0001\}$. The value of k is treated as a hyperparameter chosen in $\{4, 5\}$.

For $\Gamma_{\text{1-hot}}$, recall that we wish to learn the matrix $\boldsymbol{W}$ whose rows are subgraph representations. We choose the dimension of the representations among $\{32, 64, 128, 256\}$ and the learning rate in $\{0.001, 0.0001\}$. The value of k is treated as a hyperparameter chosen in $\{4, 5\}$.

For the GNNs, we tune the learning rate in $\{0.01, 0.001\}$, the number of hidden neurons of the MLP in Equation (15) in $\{32, 64, 128\}$, the number of layers among $\{1, 2, 3\}$.

For all these models, we use a batch size of 32 graphs and a single final linear layer with a softmax activation as the downstream classifier. We optimize for 400 epochs.

For the graph kernels, following Kriege et al. (2020), we tune the regularization hyperparameter C in SVM over the set $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. We tune the number of Weisfeiler-Lehman iterations of the WL kernel to be in $\{1, 2, 3, 4\}$ (see Kriege et al. (2020, Section 3.1)).

F.3. Erdős-Rényi Connection Probability: Size Extrapolation

We simulated Erdős-Rényi graphs (Gnp model) using NetworkX (Hagberg et al., 2008). The task is to classify the edge probability $p \in \{0.2, 0.5, 0.8\}$ of the generated graph. Table 1 shows results for a single environment task (middle), where graphs in training have all size 80, and a multiple environment task (right), where training graphs have sizes in $\{70, 80\}$ chosen uniformly at random. In both cases, the test is composed of graphs of size 140. The training, validation, and test sets are fixed. The number of graphs in training, validation, and test are 80, 40, and 100, respectively. The induced homomorphism densities are obtained for subgraphs of a fixed size $k = 5$.

For $\Gamma_{\text{1-hot}}$, we hyperparameter tune the dimension of the subgraph representations in $\{32, 64, 128, 256\}$ and the learning rate in $\{0.1, 0.01, 0.001\}$.

For the GNNs and for Γ_{GIN}, and Γ_{RPGIN}, we hyperparameter tune the number of hidden neurons in the MLP of the GNN (Equation (15)) in $\{32, 64, 128, 256\}$ (GNN is used to learn the representation for k-sized subgraph for Γ_{GIN}, and Γ_{RPGIN}). The number of layers is also a hyperparameter in $\{1, 2, 3\}$ (3 layers only for the GNNs), and the learning rate in $\{0.1, 0.01, 0.001\}$. We also hyperparameter tune the presence or absence of the Jumping Knowledge mechanism from Xu et al. (2018).

For IRM, we consider the two distinct graph sizes to be the two training environments. We tune the regularizer λ (Arjovsky et al., 2019, Section 3) in $\{4, 8, 16, 32\}$, stopping at 32 because increasing its value decreased performances.

We train all neural models for 500 epochs with batch size equal to the full training data. The downstream classifier is composed by a single linear layer with softmax activations. We perform early stopping as per Hu et al. (2020). The hyperparameter search is performed by training all models with 10 different initialization seeds and selecting the configuration that achieved the highest mean accuracy on the validation data. Then, we report the mean (and standard deviation) accuracy over the training, the validation, and the test data in Table 1 (right).

For the graph kernels, following Kriege et al. (2020), we tune the regularization hyperparameter C in SVM over the set $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. We tune the number of Weisfeiler-Lehman iterations of the WL kernel to be among $\{1, 2, 3, 4\}$ (see Kriege et al. (2020, Section 3.1)).

F.4. Extrapolation performance over SBM attributed graphs

We sample Stochastic Block Model graphs (SBM) using NetworkX (Hagberg et al., 2008). Each graph has two

blocks, having a within-block edge probability of $\boldsymbol{P}_{1,1} = \boldsymbol{P}_{2,2} = 0.2$. The cross-block edge probability is $\boldsymbol{P}_{1,2} = \boldsymbol{P}_{2,1} \in \{0.1, 0.3\}$. The label of a graph is its cross-block edge probability, i.e., $Y = \boldsymbol{P}_{1,2}$.

Vertex color distributions change with train and test environments. In training, vertices in the first block are either red or blue, with probabilities $\{0.9, 0.1\}$, respectively, while vertices in the second block are either green or yellow, with probabilities $\{0.9, 0.1\}$, respectively. In test, the probability distributions are reversed: Vertices in the first block are either red or blue, with probabilities $\{0.1, 0.9\}$, respectively, and vertices in the second block are green or yellow with probabilities $\{0.1, 0.9\}$, respectively.

Table 2 shows results for the three scenarios we considered: 1. A single environment, where training graphs are of size 20 (left), 2. A multiple environment, where training graphs have size 14 or 20, chosen uniformly at random (middle), 3. A multiple environment, where training graphs are of size 20 or 30, chosen uniformly at random (right). The test is the same in all cases, and contains graphs of size 40. The number of graphs in training, validation, and test are 80, 20, and 100, respectively. We obtain the induced homomorphism densities for Γ_{GIN}, Γ_{RPGIN}, $\Gamma_{\text{1-hot}}$ for a fixed subgraph size $k = 5$.

For the GNNs and for Γ_{GIN} and Γ_{RPGIN}, we choose the number of hidden neurons in the MLP of the GNN (Equation (15)) in $\{32, 64, 128, 256\}$, the number of layers in $\{1, 2, 3\}$ (3 layers only for the GNNs) and hyperparameter tune the presence or absence of the Jumping Knowledge mechanism from Xu et al. (2018). We add the regularization penalty in Equation (8) for Γ_{GIN} and Γ_{RPGIN} in this experiments. For Γ_{GIN} and Γ_{RPGIN}, we choose the learning rate in $\{0.01, 0.001\}$ and the regularization weight in $\{0.1, 0.15\}$. For the GNNs we choose the learning rate in $\{0.1, 0.01, 0.001\}$.

For IRM, we consider the two distict graph sizes to be the two training environments. We can not treat vertex attributes as environment here since we only have a single vertex-attribute distribution in training. We tune the regularizer λ (Arjovsky et al., 2019, Section 3) in $\{4, 8, 16, 32\}$, stopping at 32 because increasing its value decreased performances.

For $\Gamma_{\text{1-hot}}$, we hyperparameter tune the dimension of the subgraph representations in $\{32, 64, 128, 256\}$ and the learning rate in $\{0.01, 0.001\}$.

We optimize all neural models for 500 epochs with batch size equal to the full training data. We use a single layer with softmax outputs as the downstream classifier. We perform early stopping as per Hu et al. (2020). The hyperparameter search is performed by training all models with 10 different initialization seeds and selecting the configuration that achieved the highest mean accuracy on the validation data. Then, we report the mean (and standard deviation) accuracy over the training, the validation, and the test data in Table 2.

For the graph kernels, following Kriege et al. (2020), we tune the regularization hyperparameter C in SVM over the set $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. We tune the number of Weisfeiler-Lehman iterations of the WL kernel to be among $\{1, 2, 3, 4\}$ (see Kriege et al. (2020, Section 3.1)).

F.5. Extrapolation performance in real world tasks that violate our causal model

The results on graphs that violate our casusal model are reported in Table 3. We use the datasets from Morris et al. (2020), split into train, validation and test as proposed by Yehudai et al. (2021). In particular, train is obtained by considering the graphs with sizes smaller than the 50-th percentile, and test those with sizes larger than the 90-th percentile. Additionally, 10% of the training graphs is held out from training and used as validation. For statistics on the datasets and corresponding splits, see Yehudai et al. (2021).

We obtain the homomorphism densities for a fixed subgraph size $k = 4$. We observed that larger subgraph sizes, $k \geq 5$, implies a larger number of distinct subgraphs and consequently a smaller proportion of shared subgraphs in different graphs. To further reduce the number of distinct subgraphs seen by the models, we only consider the most common subgraphs in training and validation when necessary. Specifically, for NCI1 and NCI109, we only use the top 100 subgraphs (out of a total of around 300), and for DD only the 30k most common (out of a total of around 200k). For PROTEINS we keep all the distinct subgraphs (which are around 180).

For the GNNs, we follow the setup proposed in Yehudai et al. (2021), where all the GNNs have 3 layers and a final classifier composed of a feedforward neural network (MLP) with 1 hidden layer and softmax outputs. We also use a dropout of 0.3. We tune the batch size in $\{64, 128\}$, the learning rate in $\{0.01, 0.005, 0.001\}$ and the network width in $\{32, 64\}$. For Γ_{GIN} and Γ_{RPGIN}, the setup is the same, except for the number of GNN layers that is set to 2. For DD we use a fixed batch size of 256 to reduce the number of times the subgraphs are passed to the network, in order to speed up training.

For $\Gamma_{\text{1-hot}}$, we choose the batch size in $\{64, 128\}$, the learning rate in $\{0.01, 0.005, 0.001\}$ and the dimension of the subgraph representations in $\{32, 64\}$.

For IRM we tune the regularizer λ (Arjovsky et al., 2019, Section 3) in $\{8, 32, 128, 512\}$. The two environments are considered to be graphs with size smaller than the median size in the training graphs and larger than the median size in the training graphs, respectively.

Table 4. Dataset statistics, Table from Yehudai et al. (2021).

	NCI1			NCI109		
	ALL	SMALLEST 50%	LARGEST 10%	ALL	SMALLEST 50%	LARGEST 10%
CLASS A	49.95%	62.30%	19.17%	49.62%	62.04%	21.37%
CLASS B	50.04%	37.69%	80.82%	50.37%	37.95%	78.62%
NUM OF GRAPHS	4110	2157	412	4127	2079	421
AVG GRAPH SIZE	29	20	61	29	20	61

	PROTEINS			DD		
	ALL	SMALLEST 50%	LARGEST 10%	ALL	SMALLEST 50%	LARGEST 10%
CLASS A	59.56%	41.97%	90.17%	58.65%	35.47%	79.66%
CLASS B	40.43%	58.02%	9.82%	41.34%	64.52%	20.33%
NUM OF GRAPHS	1113	567	112	1178	592	118
AVG GRAPH SIZE	39	15	138	284	144	746

To mitigate the imbalance between classes in training, we reweight the classes in the loss with the training proportions for each class. We train all neural models for 1000 epochs using early stopping as per Hu et al. (2020). We test the models on the epoch achieving the highest mean Matthew Correlation Coefficient on validation because of the significant class imbalance in the test, see Table 4.

For the graph kernels, following Kriege et al. (2020), we tune the regularization hyperparameter C in SVM over the set $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. We fix the number of Weisfeiler-Lehman iterations of the WL kernel to 3 (see Kriege et al. (2020, Section 3.1)), which is comparable to the 3 GNN layers.