Learning to Count Isomorphisms with Graph Neural Networks (Technical Appendices)

Xingtong Yu,[1*] **Zemin Liu,**[2*] **Yuan Fang,**[3†] **Xinming Zhang**[1†]

[1] University of Science and Technology of China, China
[2] National University of Singapore, Singapore
[3] Singapore Management University, Singapore
yxt95@mail.ustc.edu.cn, zeminliu@nus.edu.sg, yfang@smu.edu.sg, xinming@ustc.edu.cn

A Algorithm and Complexity Analysis

Algorithm. We present the algorithm for training of Count-GNN in Alg. 1. In line 1, we initial all the parameters, as well as objective $\mathcal{L}$. In lines 3-13, we accumulate the loss for the given training tuples. In particular, in lines 4-8, we conduct the recursive edge-centric aggregation. In lines 5-7, we calculate the edge-centric representation for each edge. Then, in lines 9 and 10, we form the graph representations by aggregating all the inclusive edge-centric representations for query graph and input graph, respectively. In line 11, a counter module is employed to predict the number of subgraphs of $\mathcal{G}_i$ which are isomorphic to $\mathcal{Q}_i$. In line 12, we accumulate the loss. In line 14, we form the overall objective. Finally, in line 15 we optimize the model by minimizing objective $\mathcal{L}$.

Complexity analysis. The edge-centric aggregation increases the computation cost. Here, given a tuple $(\mathcal{Q}, \mathcal{G}, n)$, we split Count-GNN into two parts for complexity analysis, *i.e.*, edge-centric aggregation, and query-conditioned graph modulation. **(1) Edge-centric aggregation.** Supposing the average degree on $\mathcal{Q}$ and $\mathcal{G}$ is $\bar{d}$. In each edge-centric aggregation layer, each edge would access its $\bar{d}$ neighboring edges for aggregation, thus involving complexity $O(\bar{d})$. For query graph $\mathcal{Q}$ with a total of K layers, the complexity for the edge representation learning is $O(\bar{d}^K \cdot |E_\mathcal{Q}|)$. Similarly, the complexity for the edge representation learning of input graph $\mathcal{G}$ is $O(\bar{d}^K \cdot |E_\mathcal{G}|)$. **(2) Query-conditioned graph modulation.** For query graph $\mathcal{Q}$, the calculation of graph representation involves complexity of $O(|E_\mathcal{Q}|)$. For input graph $\mathcal{Q}$, it first calculates the query-conditioned modulation for all edges with a complexity of $O(|E_\mathcal{G}|)$; then the calculation of graph representation has complexity of $O(|E_\mathcal{G}|)$. The prediction w.r.t. the calculated representation of query graph and input graph has complexity of $O(1)$. In summary, the prediction for tuple $(\mathcal{Q}, \mathcal{G}, n)$ has complexity of $O(\bar{d}^K \cdot |E_\mathcal{Q}| + \bar{d}^K \cdot |E_\mathcal{G}| + |E_\mathcal{Q}| + 2|E_\mathcal{G}|)$.

[*]Co-first authors with equal contribution. Part of the work was done while at Singapore Management University.

[†]Corresponding authors.

B Proofs

We present the proofs of the theoretical results in Section 4.5 of the main paper.

Lemma 1 (Generalization). *Count-GNN can be reduced to a node-centric GNN, i.e., Count-GNN can be regarded as a generalization of the latter.* □

Proof. Without loss of generality, we consider directed graphs here, where an edge on an undirected graph can be treated as two directed edges in opposite directions. We also assume the following generic form of node-centric GNN, in which the node embedding in layer l is given by

$$\mathbf{h}_u^l = \sigma(\mathbf{W}^l(\mathbf{h}_u^{l-1} + \textsc{Aggr}(\{\mathbf{h}_i^{l-1} | \langle i, u \rangle \in E\}))), \quad (*)$$

where messages from the neighboring nodes of u in the previous layer are aggregated into $\textsc{Aggr}(\{\mathbf{h}_i^{l-1} | \langle i, u \rangle \in E\})$, which is further combined with $\mathbf{h}_u^{l-1}$, the self-information of the target node u in the previous layer. $\mathbf{W}^l$ represents the weight matrix in layer l to further map the aggregated messages from layer $l-1$ into the new representation of u in layer l. Note that different choices of the aggregation function, $\textsc{Aggr}(\cdot)$, will materialize different node-centric GNNs.

Given a directed edge $e = \langle u, v \rangle$, in the first layer of Count-GNN, we concatenate the features of the edge and its two end nodes to form the initial embedding of edge e, *i.e.*, $\mathbf{h}^0_{\langle u,v \rangle} = \mathbf{x}_u \parallel \mathbf{x}_{\langle u,v \rangle} \parallel \mathbf{x}_v \in \mathbb{R}^{d_0}$. To reduce Count-GNN into a node-centric GNN, for each edge $\langle u, v \rangle$, in the first layer we skip the features of the edge and the end node v, and only employ u's feature vector to form the initial embedding, *i.e.*, $\mathbf{h}^0_{\langle u,v \rangle} = \mathbf{x}_u$. Subsequently, it can be shown that in any layer l, the embedding of each edge $e = \langle u, v \rangle$, *i.e.*, $\mathbf{h}^l_{\langle u,v \rangle}$, is equivalent to the embedding of its start node u, *i.e.*, $\mathbf{h}^l_{\langle u,v \rangle} = \mathbf{h}^l_u$. We show the equivalence by induction as follows.

First, we already have the base case $\mathbf{h}^0_{\langle u,v \rangle} = \mathbf{h}^0_u$, since in a node-centric GNN the initial embedding of node u, $\mathbf{h}^0_u$, is simply the input features $\mathbf{x}_u$. We regard this as layer 0.

Second, we need to show the inductive step, *i.e.*, given that in layer $l-1$, $\mathbf{h}^{l-1}_{\langle u,v \rangle} = \mathbf{h}^{l-1}_u$, we can derive $\mathbf{h}^l_{\langle u,v \rangle} = \mathbf{h}^l_u$ in layer l. To obtain the edge representation $\mathbf{h}^l_{\langle u,v \rangle}$ in layer

Algorithm 1: MODEL TRAINING FOR COUNT-GNN

```
Input: Training tuples $\mathcal{T} = \{(\mathcal{Q}_i, \mathcal{G}_i, n_i)|i = 1, 2, \dots\}$, total layers number $K$, hyper-parameters $\lambda$, $\mu$.
Output: Model parameters $\Theta$.
 1: $\Theta \leftarrow$ parameters initialization, $\mathcal{L} \leftarrow 0$;
 2: while not converged do                                                        ▷ Training iteration
 3:     for each triple $(\mathcal{Q}_i, \mathcal{G}_i, n_i) \in \mathcal{T}$ do
 4:         for each layer $l \in \{1, \dots, K\}$ do
 5:             for each directed edge $\langle u, v\rangle \in E_{\mathcal{Q}_i}$ or $E_{\mathcal{G}_i}$ do
 6:                 $\mathbf{h}^l_{\langle u,v\rangle} \leftarrow \sigma(\mathbf{W}^l\mathbf{h}^{l-1}_{\langle u,v\rangle} + \mathbf{U}^l\mathbf{h}^{l-1}_{\langle \cdot,u\rangle} + \mathbf{b}^l)$;     ▷ Edge-centric aggregation, Eq. (1)
 7:             end for
 8:         end for
 9:         $\mathbf{h}_{\mathcal{Q}_i} \leftarrow \sigma(\mathbf{Q} \cdot \text{AGGR}(\{\mathbf{h}_e|e \in E_{\mathcal{Q}_i}\})$;      ▷ Query graph representation, Eq. (3)
10:         $\mathbf{h}^{\mathcal{Q}_i}_{\mathcal{G}_i} \leftarrow \sigma(\mathbf{G} \cdot \text{AGGR}(\{\tilde{\mathbf{h}}_e|e \in E_{\mathcal{G}_i}\})$;      ▷ Input graph representation, Eq. (7)
11:         $\hat{n}(\mathcal{Q}_i, \mathcal{G}_i) \leftarrow \text{RELU}(\mathbf{w}^\top \text{MATCH}(\mathbf{h}_{\mathcal{Q}_i}, \mathbf{h}^{\mathcal{Q}_i}_{\mathcal{G}_i}) + b)$;      ▷ Counter, Eq. (8)
12:         $\mathcal{L} \leftarrow \mathcal{L} + |\hat{n}(\mathcal{Q}_i, \mathcal{G}_i) - n_i|$;      ▷ Loss accumulation
13:     end for
14:     $\mathcal{L} \leftarrow \mathcal{L} + \lambda \cdot \mathcal{L}_{\text{FILM}} + \mu \cdot \|\Theta\|^2_2$;      ▷ Overall objective, Eq. (9)
15:     Update $\Theta$ by minimize $\mathcal{L}$;
16: end while
17: return $\Theta$.
```

l, we start by aggregating $\langle u, v\rangle$'s adjacent edges following Eq. (2) of the main paper, which gives

$$\begin{aligned}\mathbf{h}^{l-1}_{\langle \cdot,u\rangle} &= \text{AGGR}(\{\mathbf{h}^{l-1}_{\langle i,u\rangle}|\langle i, u\rangle \in E\}) \\ &= \text{AGGR}(\{\mathbf{h}^{l-1}_{i}|\langle i, u\rangle \in E\}).\end{aligned}$$

We further apply Eq. (1) to generate the edge representation in layer l as

$$\begin{aligned}\mathbf{h}^l_{\langle u,v\rangle} &= \sigma(\mathbf{W}^l\mathbf{h}^{l-1}_{\langle u,v\rangle} + \mathbf{U}^l\mathbf{h}^{l-1}_{\langle \cdot,u\rangle} + \mathbf{b}^l) \\ &= \sigma(\mathbf{W}^l\mathbf{h}^{l-1}_{u} + \mathbf{U}^l\text{AGGR}(\{\mathbf{h}^{l-1}_{i}|\langle i, u\rangle \in E\}) + \mathbf{b}^l)\end{aligned}$$

By constraining $\mathbf{W}^l = \mathbf{U}^l$ and fixing $\mathbf{b}^l = \mathbf{0}$, we have

$$\mathbf{h}^l_{\langle u,v\rangle} = \sigma(\mathbf{W}^l(\mathbf{h}^{l-1}_{u} + \text{AGGR}(\{\mathbf{h}^{l-1}_{i}|\langle i, u\rangle \in E\}))).$$

If we choose the same Aggr$(\cdot)$ as in Eq. (*), we get $\mathbf{h}^l_{\langle u,v\rangle} = \mathbf{h}^l_u$, where $\mathbf{h}^l_u$ is given by the node-centric GNN layer in Eq. (*).

Since the base case and inductive step both hold, we have $\mathbf{h}^l_{\langle u,v\rangle} = \mathbf{h}^l_u$ for any layer l. Recall that the base case is true if, for each edge, we remove the edge features and do not concatenate the features of its end node. Therefore, we are able to reduce Count-GNN into a node-centric GNN, which means we can regard Count-GNN as a generalization of the latter. □

Theorem 1 (Expressiveness)**.** *Count-GNN is more powerful than node-centric GNNs, which means (i) for any two non-isomorphic graphs that can be distinguished by a node-centric GNN, they can also be distinguished by Count-GNN; and (ii) there exists two non-isomorphic graphs that can be distinguished by Count-GNN but not by a node-centric GNN.* □

Proof. Assuming node-centric GNNs of the form in Eq. (*), statement (i) immediately follows from Lemma 1. Hence, we only need to show (ii), which is an existential statement that can be proven by finding an example.

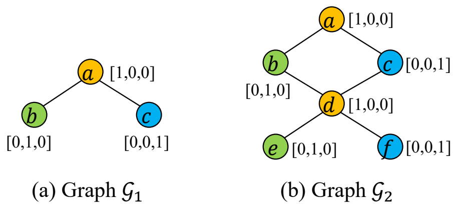


Figure I: Example graphs for Theorem 1.

The example is given in Fig. I, which consists of two non-isomorphic graphs that we need to differentiate. For instance, graph $\mathcal{G}_1$ has three nodes, *i.e.*, a, b and c, and each node has its feature vector, *e.g.*, node a has feature vector $[1, 0, 0]$. The expressiveness of GNNs determines its ability in differentiating different graph structures. In the following, we show that a node-centric GNN cannot differentiate $\mathcal{G}_1$ and $\mathcal{G}_2$ while our edge-centric Count-GNN can discriminate these two graphs. We assume both GNNs employ a mean aggregator function to aggregate the embeddings of neighboring nodes (or edges) using only one layer, and both utilize a mean readout function, *i.e.*, generating the graph representation by averaging the node (or edge) embeddings in the graph. For the ease of presentation, instead of summing the self-information and the aggregated neighboring information, we apply concatenation to fuse them.

In Tables I and II, we illustrate the calculation of node embeddings and graph embeddings using a node-centric GNN on $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. Here $\|$ is a notation for concatenation. For example, to calculate the output embedding of node a on graph $\mathcal{G}_1$, we first find the mean of its neighbors' embeddings, *i.e.*, node b with $[0, 1, 0]$ and node c with $[0, 0, 1]$, and obtain $[0.0, 0.5, 0.5]$. It is then concatenated with a's self-embedding $[1, 0, 0]$ to obtain the output embedding of a, *i.e.*, $[0.0, 0.5, 0.5 \;\|\; 1.0, 0.0, 0.0]$. It is simi-

Table I: Node-centric embeddings of graph $\mathcal{G}_1$.

	Embeddings
a	[0.0, 0.5, 0.5 ‖ 1.0, 0.0, 0.0]
b	[1.0, 0.0, 0.0 ‖ 0.0, 1.0, 0.0]
c	[1.0, 0.0, 0.0 ‖ 0.0, 0.0, 1.0]
$\mathcal{G}_1$	[0.67, 0.17, 0.17 ‖ 0.33, 0.33, 0.33]

Table II: Node-centric embeddings of graph $\mathcal{G}_2$.

	embeddings
a	[0.0, 0.5, 0.5 ‖ 1.0, 0.0, 0.0]
b	[1.0, 0.0, 0.0 ‖ 0.0, 1.0, 0.0]
c	[1.0, 0.0, 0.0 ‖ 0.0, 0.0, 1.0]
d	[0.0, 0.5, 0.5 ‖ 1.0, 0.0, 0.0]
e	[1.0, 0.0, 0.0 ‖ 0.0, 1.0, 0.0]
f	[1.0, 0.0, 0.0 ‖ 0.0, 0.0, 1.0]
$\mathcal{G}_2$	[0.67, 0.17, 0.17 ‖ 0.33, 0.33, 0.33]

Table III: Edge-centric embeddings of graph $\mathcal{G}_1$.

	embeddings
$\langle a, b\rangle$	[0.0, 0.0, 1.0, 1.0, 0.0, 0.0 ‖ 1.0, 0.0, 0.0, 0.0, 1.0, 0.0]
$\langle b, a\rangle$	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ‖ 0.0, 1.0, 0.0, 1.0, 0.0, 0.0]
$\langle a, c\rangle$	[0.0, 1.0, 0.0, 1.0, 0.0, 0.0 ‖ 1.0, 0.0, 0.0, 0.0, 0.0, 1.0]
$\langle c, a\rangle$	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ‖ 0.0, 0.0, 1.0, 1.0, 0.0, 0.0]
$\mathcal{G}_1$	[0.00, 0.25, 0.25, 0.50, 0.00, 0.00 ‖ 0.50, 0.25, 0.25, 0.50, 0.25, 0.25]

larly done for other nodes in both graphs. Finally, the graph embedding is obtained by taking the mean of the node embeddings in each graph. As shown in Tables I and II, the two non-isomorphic graphs has identical graph embeddings produced by the node-centric GNN, which means they cannot be differentiated by the node-centric GNN.

In Tables III and IV, we further illustrate the calculation of edge embeddings and graph embeddings using Count-GNN on the two graphs. Given a directed edge, we first initialize its embedding by concatenating the features of its start and end node with its own features. Since in our example the edges have no input features, we skip them in the concatenation (equivalently we can also pad zeros). For instance, for the edge $\langle d, e\rangle$ in $\mathcal{G}_2$, its initial embedding is $\mathbf{h}^0_{\langle d,e\rangle} = [1.0, 0.0, 0.0, 0.0, 1.0, 0.0]$, by concatenating d's feature vector $[1, 0, 0]$ and e's feature vector $[0, 1, 0]$. Similar to node-centric GNN, given a target edge, here we first aggregate its neighboring edges with a mean aggregator to obtain the neighboring embedding, which is then further concatenated with its self-embedding to obtain the output embedding. Again, we take the edge $\langle d, e\rangle$ on graph $\mathcal{G}_2$ as an example. We first aggregate the embeddings of its preceding edges which incident on node d, *i.e.*, $\langle b, d\rangle$, $\langle c, d\rangle$ and $\langle f, d\rangle$, thus obtaining the

Table IV: Edge-centric embeddings of graph $\mathcal{G}_2$.

	embeddings
$\langle a, b\rangle$	[0.00, 0.00, 1.00, 1.00, 0.00, 0.00 ‖ 1.00, 0.00, 0.00, 0.00, 1.00, 0.00]
$\langle b, a\rangle$	[1.00, 0.00, 0.00, 0.00, 1.00, 0.00 ‖ 0.00, 1.00, 0.00, 1.00, 0.00, 0.00]
$\langle a, c\rangle$	[0.00, 1.00, 0.00, 1.00, 0.00, 0.00 ‖ 1.00, 0.00, 0.00, 0.00, 0.00, 1.00]
$\langle c, a\rangle$	[1.00, 0.00, 0.00, 0.00, 0.00, 1.00 ‖ 0.00, 0.00, 1.00, 1.00, 0.00, 0.00]
$\langle d, b\rangle$	[0.00, 0.33, 0.67, 1.00, 0.00, 0.00 ‖ 1.00, 0.00, 0.00, 0.00, 1.00, 0.00]
$\langle b, d\rangle$	[1.00, 0.00, 0.00, 0.00, 1.00, 0.00 ‖ 0.00, 1.00, 0.00, 1.00, 0.00, 0.00]
$\langle d, c\rangle$	[0.00, 0.67, 0.33, 1.00, 0.00, 0.00 ‖ 1.00, 0.00, 0.00, 0.00, 0.00, 1.00]
$\langle c, d\rangle$	[1.00, 0.00, 0.00, 0.00, 0.00, 1.00 ‖ 0.00, 0.00, 1.00, 1.00, 0.00, 0.00]
$\langle d, e\rangle$	[0.00, 0.33, 0.67, 1.00, 0.00, 0.00 ‖ 1.00, 0.00, 0.00, 0.00, 1.00, 0.00]
$\langle e, d\rangle$	[0.00, 0.00, 0.00, 0.00, 0.00, 0.00 ‖ 0.00, 1.00, 0.00, 1.00, 0.00, 0.00]
$\langle d, f\rangle$	[0.00, 0.67, 0.33, 1.00, 0.00, 0.00 ‖ 1.00, 0.00, 0.00, 0.00, 0.00, 1.00]
$\langle f, d\rangle$	[0.00, 0.00, 0.00, 0.00, 0.00, 0.00 ‖ 0.00, 0.00, 1.00, 1.00, 0.00, 0.00]
$\mathcal{G}_2$	[0.33, 0.25, 0.25, 0.50, 0.17, 0.17 ‖ 0.50, 0.25, 0.25, 0.50, 0.25, 0.25]

Table V: Parameters for the data generation of SMALL and LARGE.

	Parameters	SMALL	LARGE
Query graph $\mathcal{Q}$	$\lvert V_{\mathcal{Q}}\rvert$	{3, 4, 8}	{3, 4, 8, 16}
	$\lvert E_{\mathcal{Q}}\rvert$	{2, 4, 8}	{2, 4, 8, 16}
	$\lvert L_{\mathcal{Q}}\rvert$	{2, 4, 8}	{2, 4, 8, 16}
	$\lvert L'_{\mathcal{Q}}\rvert$	{2, 4, 8}	{2, 4, 8, 16}
Input graph $\mathcal{G}$	$\lvert V_{\mathcal{G}}\rvert$	{8, 16, 32, 64}	{64, 128, 256, 512}
	$\lvert E_{\mathcal{G}}\rvert$	{8, 16, ..., 256}	{64, 128, ..., 2048}
	$\lvert L_{\mathcal{G}}\rvert$	{4, 8, 16}	{16, 32, 64}
	$\lvert L'_{\mathcal{G}}\rvert$	{4, 8, 16}	{16, 32, 64}

neighboring embedding $[0.00, 0.33, 0.67, 1.00, 0.00, 0.00]$. Therefore, the output edge embedding of $\langle d, e\rangle$ is [0.00, 0.33, 0.67, 1.00, 0.00, 0.00 ‖ 1.00, 0.00, 0.00, 0.00, 1.00, 0.00] after further concatenating with its initial self-embedding. Note that for edges without any preceding edge, *e.g.*, $\langle e, d\rangle$, we pad its neighboring embedding with zeros, $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]$. Finally, the graph embedding is obtained by averaging all the edge embeddings in each graph. From Tables III and IV we can observe that, the graph embeddings of $\mathcal{G}_1$ and $\mathcal{G}_2$ calculated by Count-GNN are different, which demonstrates that the two graphs can be distinguished by Count-GNN. Hence, the existential statement in (ii) is valid and the proof can be concluded. $\square$

C Details of Datasets

Data generation. We resort to the data generators in work (Liu et al. 2020) to generate the four datasets (for MUTAG and OGB-PPA, only the query graphs), by using the same parameter settings. The detailed settings in data generation for SMALL and LARGE are illustrated in Table V. In particular, when to generate one query graph or input graph, we first randomly sample the size parameters from the corresponding sets in Table V, to constrain the generation of this graph. Note that, with generally larger parameter sizes, the dataset LARGE would have larger individual graph sizes than dataset SMALL, as illustrated in Table 1.

Graph selection for OGB-PPA. The original dataset OGB-PPA consists of 158,100 graphs. However, a large fraction of graphs in OGB-PPA have no isomorphisms to the generated query graphs, and this extreme dataset distribution may impair the training of GNN-based models. Therefore, we sample 6,000 graphs from the original OGB-PPA and ensure their averaged subgraph isomorphism counting is above 10 for usage.

Query selection for secondary setting in experiments. Let N and E denote the number of nodes and directed edges, respectively; for dataset SMALL, we randomly select three query graphs in the size of (N3, E3), (N4, E4) and (N8, E8), respectively; for dataset LARGE, we randomly select three query graphs in the size of (N4, E4), (N8, E8), and (N16, E16), respectively; for dataset MUTAG, we randomly select three query graphs in the size of (N3, E2), (N4, E3) and (N4, E3), respectively.

D Details and Settings of Baselines

We compare Count-GNN with the state-of-the-art approaches from two main categories.

(1) *Conventional GNNs*, including GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), GraphSAGE (Hamilton, Ying, and Leskovec 2017), DPGCNN (Monti et al. 2018), GIN (Xu et al. 2019) and DiffPool (Ying et al. 2018). They usually capitalize on node-centric message passing, followed by a readout function to obtain the whole-graph representation.

- GCN (Kipf and Welling 2017): GCN usually resorts to mean-pooling based node-centric neighborhood aggregation to receive messages from the neighboring nodes for node representation learning.
- GAT (Veličković et al. 2018): GAT also depends on node-centric neighborhood aggregation for node representation learning, while it can assign different weights to neighbors to reweight their contributions.
- GraphSAGE (Hamilton, Ying, and Leskovec 2017): GraphSAGE has a similar neighborhood aggregation mechanism with GCN, while it focuses more on the information from the node itself.
- DPGCNN (Monti et al. 2018): DPGCNN is also a node-centric GNN models, and employs a similar neighborhood aggregator with GAT, while it also takes edge topology into account when calculating the neighbors' weights.
- GIN (Xu et al. 2019): GIN employs a SUM aggregator to replace the mean-pooling method in GCN to aggregate all the messages from neighboring nodes, which is demonstrated to be more powerful to capture the graph structures.
- DiffPool (Ying et al. 2018): DiffPool depends on a GNN framework to further build its specific aggregation mechanism, by clustering nodes hierarchically to form the whole-graph representation.

(2) *GNN-based isomorphism counting models*, including four variants proposed by (Liu et al. 2020), namely RGCN-DN, RGCN-Sum, RGIN-DN, RGIN-Sum, as well as LRP (Zhengdao et al. 2020) and DMPNN-LRP (Liu and Song 2022). They are purposely designed GNNs for subgraph isomorphism counting, relying on different GNNs (*e.g.*, RGCN (Schlichtkrull et al. 2018), RGIN (Xu et al. 2019), or local relational pooling (Zhengdao et al. 2020)) for node representation learning, followed by a specialized readout suited for isomorphism matching, *e.g.*, DiamNet (Liu et al. 2020). In particular, the two variants RGCN-DN and RGIN-DN utilize DiamNet, whereas RGCN-Sum and RGIN-Sum utilize the simple sum-pooling. Based on previous work (Liu et al. 2020; Zhengdao et al. 2020), DMPNN (Liu and Song 2022) also leverage edge-centric aggregation via dual graph. DMPNN-LRP (Liu and Song 2022) adds local relational pooling behind dual message passing for node representation learning compared with DMPNN.

Model settings. To achieve the optimal performance, we tune the hyper-parameters for all the baselines according to the proposed settings in literature. In particular, for conventional GNN models including GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), GraphSAGE (Hamilton, Ying, and Leskovec 2017), DPGCNN (Monti et al. 2018), GIN (Xu et al. 2019) and DiffPool (Ying et al. 2018), we set the number of total layers as 3, hidden dimension as 128, and dropout rate as 0.2. In particular, for GAT, we use a self-attention mechanism with 4 heads; for GraphSAGE, we use the mean-pooling as the aggregator; for DPGCNN, we use 4 heads on the dual convolution layer and set the output dimension as 8 for each head. For DiffPool, we set the ratio of nodes' number in consecutive layers as 0.1. For GNN-based isomorphism counting models, we follow the hyper-parameter settings in their original papers, with which the models can achieve the optimal performance. In particular, for RGCN-SUM, RGCN-DM, RGIN-SUM ,RGIN-DM and DMPNN-LRP we set the number of layers as 3, and the hidden dimension as 128.

E Settings of Count-GNN

We tune several hyper-parameters for Count-GNN to achieve its optimal performance. In particular, we employ a Count-GNN with a total of 3 layers. Besides, on SMALL and LARGE datasets, we set the hidden dimension as 24, due to the fact that Count-GNN performs well even with low hidden dimensions, though it usually performs better with higher dimension which also costs more time. To find a balance between the accuracy and time cost, we choose this moderate dimension. On MUTAG, we set the hidden dimension as 12. On OGB-PPA, we set the hidden dimension as 24. In addition, we set the hyper-parameter λ for weighting the FiLM factors in Eq. (9) as 0.0001.

F Further Experiments and Analysis

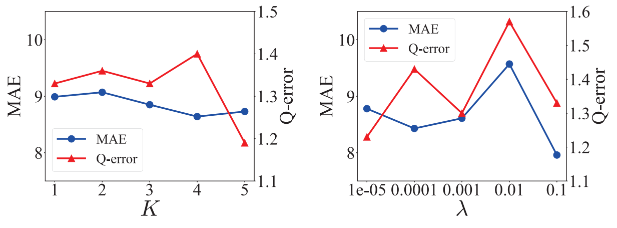
In addition to the experiments in the main paper, in this section we present further experimental results and analysis for model evaluation.

Parameters Sensitivity. We study the sensitivity of two important hyper-parameters on SMALL dataset.

In Fig. II(a), we increase the total number of GNN layers K for edge-centric aggregation, to check its influence on the performance. As K increases, the performance in terms of MAE and Q-error generally become better, only with one exception on Q-error when $K = 4$. This shows a phenomena that the increase of layers may facilitate the exploitation of long-range structural information, which might further help the model to achieve a clearer view of the graph structure.

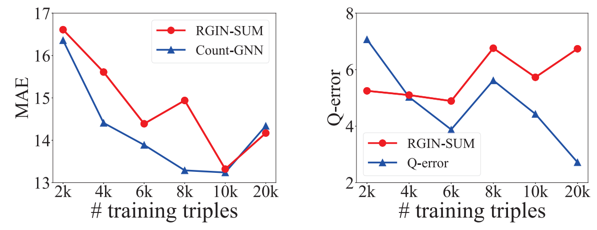
In Fig. II(b), we show the sensitivity of parameter λ, which weights the regularizer on the FiLM factors in Eq. (9). We observe that λ is relative sensitive to the performance, and $\lambda = 0.01$ may result in an inferior performance. Interval [1e-5, 1e-3] might be a good range for superior performance of subgraph isomorphism counting.

Comparison with Different Training Sizes. To evaluate the performance tendency of Count-GNN with different training sizes, we conduct an experiment by increasing the number of training triplets from 2,000 to 10,000, then to 20,000 on dataset SMALL. A baseline RGIN-SUM (Liu et al. 2020) is also employed for comparison, which



(a) Sensitivity of $K \downarrow$ (b) Sensitivity of $\lambda \downarrow$

Figure II: Parameters sensitivity on dataset SMALL.



(a) MAE $\downarrow$ (b) Q-error $\downarrow$

Figure III: Comparison with different training sizes.

is proved to be competitive in the results of Table (2). Figs. III(a) and III(b) show the results of MAE and Q-error, respectively. We have the following observations. First, with different training sizes, the proposed model Count-GNN can consistently outperform baseline RGIN-SUM. The only exceptions lie in MAE with 20k and Q-error with 2k. This demonstrates that the performance of Count-GNN for subgraph isomorphism counting is stably superior to the baselines with different sizes of labeled data. Only when labeled data is too scarce or too sufficient its performance might be surpassed by the competitive baselines. Second, as the number of training triplets increases, both MAE and Q-error have a tendency of decrease, showing that more labeled data would generally boost the model performance.

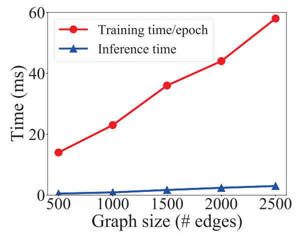


Figure IV: Scalability study.

Scalability study. We investigate the scalability of Count-GNN on the dataset OGB-PPA, which contains graphs with the largest average number of edges. For both training and testing, we first randomly sample 10 query graphs. Next, we construct five groups of input graphs, where each group contains 10 input graphs of similar size in terms of number of edges. The average number of edges per graph in the five group ranges between 500 and 2500. We illustrate both the training and inference time (in ms) for each group in Fig. IV. Note that the time costs reported here are much smaller than the numbers in Tables 2 and 3, as there are only 10 query/input graphs per group. Both training and inference time increase linearly w.r.t. the number of edges in the graph. The linear growth demonstrates that Count-GNN is capable of scaling to larger and denser graphs.

G Additional Related Work

Graph representation learning. Graph representation learning (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016; Tang et al. 2015) usually capitalizes on substructures sampling on graph to represent the local view of graph structures, thus an encoder can be further employed to embed nodes into low-dimensional representations, in which the graph structures are preserved. More recently, graph neural networks (GNNs) (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Veličković et al. 2018; Xu et al. 2019) arise as a powerful family of representation learning approaches, which rely on the key operator of neighborhood aggregation to pass messages recursively for node representation learning, thus both the structure and content information can be preserved simultaneously.

Other related studies. Graph similarity search (Bai et al. 2019; Li et al. 2019) addresses a related but distinct problem of evaluating the similarity between two graphs. Some recent studies (Wang, Yan, and Yang 2019; Wang et al. 2021; Bai et al. 2021) also attempt to combine the traditional models and deep learning models. Subgraph similarity search (Yuan et al. 2012) is another similar yet different task, which aims to calculate whether a target graph approximately contains a query graph. Many of them capitalize on Subgraph Edit Distance (SED) to calculate the similarity between the target and query graphs (Bai et al. 2020; Zhang et al. 2021). However, both of the prior two search methods cannot be directly employed to cope with the problem of subgraph isomorphism counting due to the difference in problem. Object detection (Redmon et al. 2016; Zhao et al. 2019), which is intrinsically similar to subgraph isomorphism counting on graph, is a popular topic in the field of computer vision. However, due to the divergent data characteristics between graph and visual data, object detection approaches cannot be applied to solve subgraph isomorphism counting on graphs. Separately, GSN (Bouritsas et al. 2020) employs a topology-aware message passing scheme, in which substructure isomorphism counts are used as structural features to enhance the expressive power of GNNs. That is, it simply applies an existing algorithm for subgraph isomorphism counting and leverages the counts as additional input, but does not address the problem of subgraph isomorphism counting itself.

References

Bai, Y.; Ding, H.; Bian, S.; Chen, T.; Sun, Y.; and Wang, W. 2019. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 384–392.

Bai, Y.; Ding, H.; Gu, K.; Sun, Y.; and Wang, W. 2020. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3219–3226.

Bai, Y.; Xu, D.; Sun, Y.; and Wang, W. 2021. GLSearch: Maximum Common Subgraph Detection via Learning to Search. In *International Conference on Machine Learning*, 588–598. PMLR.

Bouritsas, G.; Frasca, F.; Zafeiriou, S.; and Bronstein, M. M. 2020. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*, 855–864.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NIPS*, 1024–1034.

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

Li, Y.; Gu, C.; Dullien, T.; Vinyals, O.; and Kohli, P. 2019. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, 3835–3845. PMLR.

Liu, X.; Pan, H.; He, M.; Song, Y.; Jiang, X.; and Shang, L. 2020. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1959–1969.

Liu, X.; and Song, Y. 2022. Graph convolutional networks with dual message passing for subgraph isomorphism counting and matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 7594–7602.

Monti, F.; Shchur, O.; Bojchevski, A.; Litany, O.; Günnemann, S.; and Bronstein, M. M. 2018. Dual-primal graph convolutional networks. *arXiv preprint arXiv:1806.00770*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: Online learning of social representations. In *KDD*, 701–710.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*, 593–607. Springer.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*, 1067–1077.

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

Wang, R.; Yan, J.; and Yang, X. 2019. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 3056–3065.

Wang, R.; Zhang, T.; Yu, T.; Yan, J.; and Yang, X. 2021. Combinatorial learning of graph edit distance via dynamic embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5241–5250.

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

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *Advances in Neural Information Processing Systems*, 31: 4800–4810.

Yuan, Y.; Wang, G.; Chen, L.; and Wang, H. 2012. Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. *Proceedings of the VLDB Endowment*, 5(9).

Zhang, Z.; Bu, J.; Ester, M.; Li, Z.; Yao, C.; Yu, Z.; and Wang, C. 2021. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2274–2284.

Zhao, Z.-Q.; Zheng, P.; Xu, S.-t.; and Wu, X. 2019. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11): 3212–3232.

Zhengdao, C.; Lei, C.; Soledad, V.; and Bruna, J. 2020. Can Graph Neural Networks Count Substructures? *Advances in neural information processing systems*.