A APPENDIX

Theorem A.1. Given a target graph $G_{\mathcal{T}}(V_{\mathcal{T}}, E_{\mathcal{T}})$ and a query graph $G_{\mathcal{Q}}(V_{\mathcal{Q}}, E_{\mathcal{Q}})$, if $G_{\mathcal{Q}} \subset G_{\mathcal{T}}$, and the subtree generation function Ψ as defined in Eq. (2) meets the following condition:

$$\forall \text{ graph pair } (G_{\mathcal{S}}, G), \text{if } G_{\mathcal{S}} \subset G, \text{then } \Psi(G_{\mathcal{S}}) \subset \Psi(G), \tag{9}$$

then there exists an injective function $f : V_{\mathcal{Q}} \to V_{\mathcal{T}}$, ensuring the l-hop subtrees of the subgraph is isomorphic to the subtrees of the corresponding subgraph:

$$\forall l \geq 1, \mathrm{q} \in V_{\mathcal{Q}}, \mathrm{t} = f(\mathrm{q}) \in V_{\mathcal{T}} \Rightarrow T_{\mathrm{q}}^{(l)} \subset T_{\mathrm{t}}^{(l)}, \tag{10}$$

Proof. According to the definition of subgraph matching (McCreesh et al., 2018), when $G_{\mathcal{Q}}$ is a subgraph of $G_{\mathcal{T}}$, there must exists an injective function $f : V_{\mathcal{Q}} \to V_{\mathcal{T}}$, such that $\forall \mathrm{q}_i, \mathrm{q}_j \in V_{\mathcal{Q}}, (\mathrm{q}_i, \mathrm{q}_j) \in E_{\mathcal{Q}} \Rightarrow (f(\mathrm{q}_i), f(\mathrm{q}_j)) \in E_{\mathcal{T}}$. For any subgraph in the query graph, e.g., $S(V_S, E_S) \in G_{\mathcal{Q}}$, we always have a subgraph in the original graph $G_{\mathcal{T}}$, denoted as $G_S(V_G, E_G)$, that corresponds to the set of the query node as $V_G = f(V_S)$. This tells us that $S \subset G_S$. According to this, consider any given node from $V_{\mathcal{Q}}$: $\mathrm{q} \in V_{\mathcal{Q}}$, $S_{\mathrm{q}}^{(l)}$ is a subgraph of $G_{\mathcal{Q}}$ and its image $G_{S_{\mathrm{q}}^{(l)}}$ in $G_{\mathcal{T}}$, i.e. $S_{\mathrm{q}}^{(l)} \subset G_{S_{\mathrm{q}}^{(l)}}$. By definition, the node in $S_{\mathrm{q}}^{(l)}$ or $G_{S_{\mathrm{q}}^{(l)}}$ is at most l-hop from node q or $\mathrm{t} = f(\mathrm{q})$, we know that $G_{S_{\mathrm{q}}^{(l)}}$ must be a subgraph of $S_{\mathrm{t}}^{(l)}$, i.e.,$G_{S_{\mathrm{q}}^{(l)}} \subset S_{\mathrm{t}}^{(l)}$. Put all together, we have $S_{\mathrm{q}}^{(l)} \subset G_{S_{\mathrm{q}}^{(l)}} \subset S_{\mathrm{t}}^{(l)}$. Based on the listed constrain, we then have $T_{\mathrm{q}}^{(l)} \subset T_{\mathrm{t}}^{(l)}$.

□

Theorem A.2. Given a node q in the query graph and a node t in the target graph, the following three conditions are equivalent:

1) $T_{\mathrm{q}}^{(l+1)} \subset T_{\mathrm{t}}^{(l+1)}$.
2) There exists an injective function on the neighborhood of these nodes as $f : N(\mathrm{q}) \to N(\mathrm{t})$, s.t. $\forall \mathrm{q}_i \in N(\mathrm{q}), \mathrm{t}_i = f(\mathrm{q}_i), T_{\mathrm{q}_i}^{(l)} \subset T_{\mathrm{t}_i}^{(l)}$.
3) There exists a perfect matching on the bipartite graph $B^{(l)}(N(\mathrm{t}), N(\mathrm{q}), E)$, where $\forall \mathrm{t}_j \in N(\mathrm{t}), \mathrm{q}_i \in N(\mathrm{q}), (\mathrm{t}_j, \mathrm{q}_i) \in E$ if and only if $T_{\mathrm{q}_i}^{(l)} \subset T_{\mathrm{t}_j}^{(l)}$.

We prove this theorem by introducing the following two theorem. Theorem A.3 shows that condition 1) is equivalent to condition 2), i.e. the WL subtree isomorphism test can be accomplished in a recursive manner then prove Theorem. A.4 that the condition 2) equals to condition 3) which means every iteration in the recursive process equals to examine the existence of a perfect matching, respectively.

Theorem A.3. Given a node q in the query graph and a node t in the target graph, the following two conditions are equal:
1) $T_{\mathrm{q}}^{(l+1)} \subset T_{\mathrm{t}}^{(l+1)}$, where l is an integer and $l \geq 1$.
2) There exists an injective function on the neighboring set of these nodes as $f : N(\mathrm{q}) \to N(\mathrm{t}), \text{s.t.} \forall \mathrm{q}_i \in N(\mathrm{q}), \mathrm{t}_i = f(\mathrm{q}_i), T_{\mathrm{q}_i}^{(l)} \subset T_{\mathrm{t}_i}^{(l)}$.

Proof. We assume f_{q} is a subtree isomorphism injective function in the condition 1), that $\forall$ node $u, v \in T_{\mathrm{q}}^{(l+1)}, (u, v)$ is an edge of $T_{\mathrm{q}}^{(l+1)} \Rightarrow ((f_{\mathrm{q}}(u), f_{\mathrm{q}}(v))$ is an edge of $T_{\mathrm{t}}^{(l+1)}$. Similarly We also assume f_{q_i} is subtree isomorphism injective in the condition 2).
On the one hand, if condition 1) is true then f_{q} exists. Using the property of WL tree, we have $\forall \mathrm{q}_i \in N(\mathrm{q}), T_{\mathrm{q}_i}^{(l)} \subset T_{\mathrm{q}}^{(l+1)}$, which means the l-order WL tree of any node q_i in q's neighbourhood belongs to the $l+1$-order WL tree originate from the node q. This suggests that f_{q} maps $T_{\mathrm{q}_i}^{(l)}$ into a tree $T_{f(\mathrm{q}_i)}^{(l)} = T_{\mathrm{t}_i}^{(l)}$, which is a subtree of $T_{\mathrm{t}}^{(l+1)}$,. Then the condition 2) is true.
On the other hand, if condition 2) holds, then we define the mapping as $f_{\mathrm{q}}(v) = \begin{cases} f_{\mathrm{q}_i}(v), & v \in T_{\mathrm{q}_i}^{(l)} \\ \mathrm{q}, & v = \mathrm{q} \end{cases}$.

Here, the function $f_b(v)$ is a standard injective function $T^{(l)}_{b_i}$ This implies this is a subtree isomorphic mapping, so 1) holds.

□

The above theorem provides a recursive solution to the WL subtree isomorphism algorithm. Intuitively, we can maintain an indicator matrix $S^{(l)} \in R^{|V_\mathcal{T}|\times|V_\mathcal{Q}|}$, where $S^{(l)}_{\mathrm{tq}} = \begin{cases} 1, & T^{(l)}_{\mathrm{q}} \subset T^{(l)}_{\mathrm{t}} \\ 0, & \text{else} \end{cases}$. This matrix captures the relation between all pairs of nodes and thus can be used for recursion update. Next, we will show that the update process can be implemented as a perfect matching problem, i.e., what makes condition 2) true is equivalent to finding a perfect matching on a bipartite graph, as shown in the following theorem:

Theorem A.4. Assume the neighboring set of node t and q as $X = N(\mathrm{t})$ and $Y = N(\mathrm{q})$, respectively. Accordingly, we form a bipartite graph as $B^{(l)}_{\mathrm{t,q}}(X, Y, E)$. Here, we define the edges as $E = \{(\mathrm{t}_i, \mathrm{q}_j) : T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_j}\}$, where t_i and q_j represent the ith and jth neighbour of node t and q, respectively. Under this setting, the injective function f from the condition 2) in Theorem. A.3 induces a perfect matching.

Proof. The injective function f of condition 2) in Theorem A.3 maps every node q_i in $N(\mathrm{q})$ to $\mathrm{t}_i = f(\mathrm{q}_i) \in N(\mathrm{t})$ and $T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_i}$ holds. While $T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_i}$ means $(\mathrm{q}_i, \mathrm{t}_i) \in E$, the injective f naturally corresponds every node q_i to an edge $(\mathrm{q}_i, \mathrm{t}_j)$. Since f is an injective function, $\mathrm{q}_{i_1} \neq \mathrm{q}_{i_2} \Rightarrow \mathrm{t}_{i_1} \neq \mathrm{t}_{i_2}$, indicating that all these edges $(\mathrm{q}_i, \mathrm{t}_i), i = 1, ..., |N(\mathrm{q})|$ are different, which actually forms a perfect matching. □

Theorem A.5. Given the sampled query graph and the target graph, we can construct their adjacency matrices , $\tilde{A}_\mathcal{Q}$ and $A_\mathcal{T}$, and the degree matrix of the sampled query graph $\tilde{D}_\mathcal{Q} = \mathrm{diag}(\sum_s((\tilde{A}_\mathcal{Q})_{:s}))$. Here, we denote the indicator matrix at the l-th hop as $S^{(l)}$. To check the validity of $|N(W)| \geq |W|$, we can check whether each element of Φ is true or not, where $\Phi \coloneqq Z_{N(W)} \geq 1$, $Z_{N(W)} = \mathrm{aggregate}_{\mathrm{sum}}(A_\mathcal{T}, Z^T_W)$ and $Z_W = \mathrm{aggregate}_{\mathrm{max}}(\tilde{D}^{-1}_\mathcal{Q} \cdot \tilde{A}_\mathcal{Q}, (S^{(l)})^T)$.

Proof. For each node pair t, q and their corresponding $W = N'(\mathrm{q})$ in the sampled query graph, We first transform the neighboring set of W, i.e., $N(W)$, as following:

$$\begin{aligned} N(W) &= \{\mathrm{t}_i \in N(\mathrm{t}) | \exists \mathrm{q}_j \in W = N'(\mathrm{q}), \text{s.t.} T^{(l)}_{\mathrm{q}_j} \subset T^{(l)}_{\mathrm{t}_i}\} \\ &= \{\mathrm{t}_i \in N(\mathrm{t}) | \exists \mathrm{q}_j \in W = N'(\mathrm{q}), \text{s.t.} S_{\mathrm{t}_i,\mathrm{q}_j} = 1\} \\ &= \{\mathrm{t}_i \in N(\mathrm{t}) | \max_{\mathrm{q}' \in N'(\mathrm{q})} S^{(l)}_{\mathrm{t}_i,\mathrm{q}'} = 1\} \\ &= N(\mathrm{t}) \cap \{\mathrm{t}_i | \max_{\mathrm{q}' \in N'(\mathrm{q})} S^{(l)}_{\mathrm{t}_i,\mathrm{q}'} = 1\} \\ &= N(\mathrm{t}) \cap M(\mathrm{q}) \end{aligned} \tag{11}$$

Let $M(\mathrm{q}) = \{\mathrm{t}_i | \max_{\mathrm{q}' \in N'(\mathrm{q})} S^{(l)}_{\mathrm{t}_i,\mathrm{q}'} = 1\}$, we can compute $M(\mathrm{q})$ via a standard maximizing aggregation process on the sampled adjacency matrix $\tilde{A}_\mathcal{Q}$, in which treats the indicator matrix $(S^{(l)})^T \in R^{|V_\mathcal{Q}|\times|V_\mathcal{T}|}$ as node attributes. This process will output the representation of node q as follows,

$$z_{\mathrm{q},:} = max\{(S^{(l)})^T_{j,:}, \forall j \in N'(\mathrm{q})\}, \tag{12}$$

The obtained vector $z_{\mathrm{q},:}$ is to represent $M(\mathrm{q})$ where $z_{\mathrm{q}i} = \begin{cases} 1, & i \in M(\mathrm{q}) \\ 0, & \text{else} \end{cases}$. We rewrite this into a matrix format as

$$Z_W = \mathrm{aggregate}_{\mathrm{max}}(\tilde{A}_\mathcal{Q}, (S^{(l)})^T) \tag{13}$$

where $Z_W \in R^{|V_\mathcal{Q}|\times|V_\mathcal{T}|}$ and its q-th row vector is $z_{\mathrm{q}:}$.

Recall that we demand $N(W) = N(\mathrm{t}) \cap M(\mathrm{q})$. After acquiring $M(\mathrm{q})$, we can compute the $|N(W)|$ as follows,

$$\begin{cases} |M(\mathrm{q})| = \sum_i z_{\mathrm{q},i} \\ |N(W)| = \sum_i z_{\mathrm{q},i}, i \in N(\mathrm{t}) \end{cases} \tag{14}$$

In essence, this is to implement a summation aggregation on the target graph using the node representation Z_W, i.e.,

$$Z_{N(W)} = \text{aggregate}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T) \tag{15}$$

where $Z_{N(W)} \in R^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$ is an integer matrix and its element (t, q) shows the score of $|N(W)|$ between node t and q. This transformation converts the counting operation as aggregation such that we can check the aggregated values to determine whether there is a perfect matching. Given a node pair (t, q), we have $|N(W)| = [Z_{N(W)}]_{\mathrm{tq}}$ and $|W| = |N'(\mathrm{q})| = \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s}$. Therefore, the question becomes to check whether $[Z_{N(W)}]_{\mathrm{tq}} \geq \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s}$ holds. We can then derive the perfect matching as follows:

$$\begin{aligned} & [Z_{N(W)}]_{\mathrm{tq}} \geq \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s} \\ \Leftrightarrow & [Z_{N(W)}]_{\mathrm{tq}} / \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s} \geq 1 \\ \Leftrightarrow & [\text{aggregate}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T)]_{\mathrm{tq}} / \tilde{d}_{\mathrm{q}} \geq 1 \\ \Leftrightarrow & [\text{aggregate}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T) \cdot \tilde{D}_{\mathcal{Q}}^{-1}]_{\mathrm{tq}} \geq 1 \\ \Leftrightarrow & [\text{aggregate}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T \cdot \tilde{D}_{\mathcal{Q}}^{-1})]_{\mathrm{tq}} \geq 1 \end{aligned} \tag{16}$$

where $\tilde{d}_{\mathrm{q}}$ is the degree of node q in the sampled graph. The degree matrix of the sample graph is defined as $\tilde{D}_{\mathcal{Q}} = \text{diag}[\sum_s ((\tilde{A}_{\mathcal{Q}})_{:s})]$. Now recall that Φ is the matrix whose (t, q) element is the comparison result of $|N(W)|$ and $|W|$ of (t, q), according to eq 16, we have:

$$\Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}, A_{\mathcal{T}}) = \text{aggregate}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T \cdot \tilde{D}_{\mathcal{Q}}^{-1}) \geq 1, \tag{17}$$

where

$$\begin{aligned} Z_W^T \cdot \tilde{D}_{\mathcal{Q}}^{-1} &= [\text{aggregate}_{\max}(\tilde{A}_{\mathcal{Q}}, (S^{(l)}))]^T \cdot \tilde{D}_{\mathcal{Q}}^{-1} \\ &= [\tilde{D}_{\mathcal{Q}}^{-1} \cdot \text{aggregate}_{\max}(\tilde{A}_{\mathcal{Q}}, (S^{(l)}))]^T \\ &= [\text{aggregate}_{\max}(\tilde{D}_{\mathcal{Q}}^{-1} \cdot \tilde{A}_{\mathcal{Q}}, (S^{(l)}))]^T \end{aligned} \tag{18}$$

□

Theorem A.6. Every chordless cycle is atomic. Every chordless cycle $\mathcal{C}_{\mathcal{Q}}$ in an induced subgraph $G_{\mathcal{Q}}$ must correspond to a chordless cycle $\mathcal{C}_{\mathcal{T}}$ in the origin graph $G_{\mathcal{T}}$.

Proof. Chordless cycle does not have any chord, thus there is no smaller cycle in the chordless cycle, which means chordless cycle is atomic. Assuming $G_{\mathcal{Q}}$ is a subgraph of $G_{\mathcal{T}}$, every node of $\mathcal{C}_{\mathcal{Q}}$ must correspond to a node in $G_{\mathcal{T}}$, and these nodes form a circle $\mathcal{C}_{\mathcal{T}}$ in $G_{\mathcal{T}}$. Since $G_{\mathcal{Q}}$ is an induced subgraph of $G_{\mathcal{T}}$, if $\mathcal{C}_{\mathcal{T}}$ has a chord, then $\mathcal{C}_{\mathcal{Q}}$ must have a chord, which contradicts the condition that $\mathcal{C}_{\mathcal{Q}}$ is a chordless graph. □

A.1 Implementation Details

The python implementation of D^2Match is available at:

`https://www.dropbox.com/sh/8pvj8drvj0l2zou/AAB5j7e7frVwMiun1QcCNbMFa?dl=0`

At the beginning of subtree isomorphism test, the model needs an initial indicator matrix $S_{subtree}^{(0)}$ as the input of the first iteration. According to the definition of the indicator matrix, $S_{subtree}^{(0)}$ shows the isomorphism relation between the subtree of 0-hop neighbors, which are the nodes themselves in this case. Since all nodes will be isomorphic to each other if not considering the node attributes, the indicator matrix $S_{subtree}^{(0)}$ is actually a similarity matrix w.r.t node attributes. To get a similarity matrix of attributes, we can either directly calculate the similarity between nodes or employ neural

networks on these attributes to learn the matrix. In our model, we implement both methods to initialize the matrix, called the initialization of the raw and the learnable:

$$\begin{aligned} Raw : S^{(0)}_{subtree} &= CosineSimilarity(X_{\mathcal{T}}, X_{\mathcal{Q}}) = Norm(X_{\mathcal{T}}) \cdot Norm(X^T_{\mathcal{Q}}) \\ Learnable : S^{(0)}_{subtree} &= MLP(X_{\mathcal{T}}) \cdot MLP(X_{\mathcal{Q}})^T \end{aligned} \tag{19}$$

where the raw initialization is to calculate the cosine similarity between the nodes' attributes, and the learnable initialization employs a MLP to generate hidden representations of nodes and compute their dot similarities.

In practice, we find the raw initialization performs better. This is because the node attributes of datasets are usually binary categorical vectors, which induces clear identification information of the nodes and can be easily captured by cosine similarity.

Our implementation of the GNN block in the model is slightly different from the description. Specifically, we use compute the similarity of each pair of nodes as:

$$[S^{(l+1)}_{gnn}]_{ij} = MLP(concat([H^{(l)}_{\mathcal{T}}]_i, [H^{(l)}_{\mathcal{Q}}]_j)). \tag{20}$$

The main difference is that we do not output a $|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|$ matrix, but a $|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}| \times |D^{(l+1)}|$ tensor, where $D^{(l+1)}$ denotes the hidden dim of $l+1$ layer. The intuition is that a tensor that represents the node pairs' similarity with vectors can retain more information than a similarity matrix with scalar elements. In this setting, the final indicator matrix $S^{(l+1)}$ can not be generated as $S^{(l+1)} = S^{(l+1)}_{gnn} \odot S^{(l+1)}_{subtree}$, because $S^{(l+1)}_{subtree} \in R^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$ but $S^{(l+1)}_{gnn} \in R^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}| \times |D^{(l+1)}|}$. Thus we broadcast $S^{(l+1)}_{subtree}$ to $\tilde{S}^{(l+1)}_{subtree}$ where $\forall k \in [0, D^{(l+1)}), [\tilde{S}^{(l+1)}_{subtree}]_{ijk} = [S^{(l+1)}_{subtree}]_{ij}$ and the final indicator matrix $S^{(l+1)} = S^{(l+1)}_{gnn} \odot \tilde{S}^{(l+1)}_{subtree}$

At the end of our models, we get the subtree indicator matrix $S^{(L)}_{subtree}$ and the GNN indicator matrix $S^{(L)}_{gnn}$. The model will output the final score from $S^{(L)}_{subtree}$ and $S^{(L)}_{gnn}$, respectively. For the subtree module, we check whether the indicator matrix is feasible to induce the subgraph isomorphism. Note that for a node i in the target graph and a node j in the query graph, i is possible to match j unless $[S^{(L)}_{subtree}]_{ij} = 1$. So we check whether the subtree indicator matrix meets the following two conditions:

1) Every node in a query graph should match at least one node in the target graph:

$$\begin{aligned} &\forall j, \max_i (S^{(L)}_{subtree})_{ij} = 1 \\ \Leftrightarrow &\sum_j \max_i (S^{(L)}_{subtree})_{ij} = |V_{\mathcal{Q}}| \\ \Leftrightarrow &\sum_j \max_i (S^{(L)}_{subtree})_{ij} / |V_{\mathcal{Q}}| = 1 \end{aligned} \tag{21}$$

2) The number of nodes in the target graph that match at least one node in the query graph is more than the number of nodes of query graph:

$$\sum_i \max_j (S^{(L)}_{subtree})_{ij} \geq |V_{\mathcal{Q}}| \Leftrightarrow \quad \sum_i \max_j (S^{(L)}_{subtree})_{ij} / |V_{\mathcal{Q}}| \geq 1 \tag{22}$$

To make the subtree model differentiable, we use a learnable sigmoid to replace all the logical judgment in the model:

$$LSigmoid(x) = \sigma(ax + b) \tag{23}$$

where a, b are learnable parameters; σ is the sigmoid function. The result of subtree module can be fomulated as:

$$r_{subtree} = LSigmoid(\sum_i \max_j (S^{(L)}_{subtree})_{ij} / |V_{\mathcal{Q}}|) \cdot LSigmoid(\sum_j \max_i (S^{(L)}_{subtree})_{ij} / |V_{\mathcal{Q}}|) \tag{24}$$

For the GNN module, we employ the neural tensor network(NTN) (Bai et al., 2019) and generate a score according to the output of NTN and the aggregated indicator tensor:

$$r_{gnn} = \sigma(MLP(concat[NTN(H^{(L)}_{\mathcal{T}}, H^{(L)}_{\mathcal{Q}}), \sum_i \sum_j S^{(L)}_{subtree}])) \tag{25}$$

Where $H_{\mathcal{T}}^{(L)}, H_{\mathcal{Q}}^{(L)}$ are the node representations generated by the GNNs. NTN is the NTN layer.

The final prediction is:

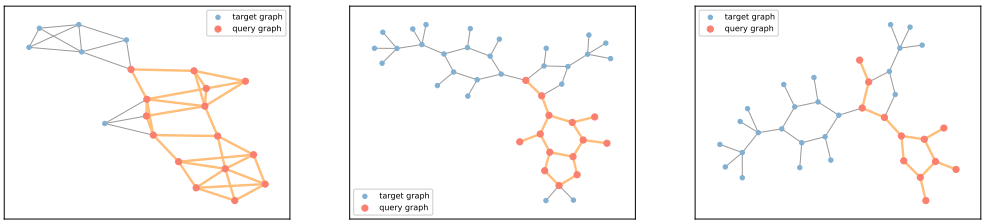
$$r = r_{gnn} \cdot r_{subtree} \tag{26}$$

Although the model's prediction is obtained by integrating the two modules, we can not directly train the model through the final score r because it will bring difficulties in the training process. When fitting a negative sample, the resulting subtree module tends to be zero, forcing the overall gradient to be zero which hinders the training of the GNN module.

Therefore, we train the two blocks with different objectives. For the subtree module which aims to learn the isomorphism relation, the result should be either 0 for not matching or 1 for matching. So we employ MAE loss to enforce the results to be either 0 or 1. For the GNN module, we use MSE to encourage the output of GNNs to capture the similarity. Suppose the ground-truth label is y, and our loss function is

$$L = MSE(r_{gnn}, y) + MAE(r_{subtree}, y) \tag{27}$$

Both our model and all baselines use the Adam as optimizer and set the learning rate to $3e-4$. To ensure fairness, we set all models with adjustable number of layers to 5 layers, and set the hidden dimension to 10.



(a) (b) (c)

Figure 3: The detected subgraphs by D^2Match

A.2 D^2MATCH AT WORK

Recall that D^2Match learns an indicator matrix to capture pairwise similarities. It plays the role of permutation matrix in matching, allowing us to pinpoint the matched subgraph. This is particularly useful since the exact position is required for some downstream applications such as web search. In comparison, other learning-based methods are unable to pinpoint local correspondences, but only establish the existence of a matching. We provide a visualization of the matched subgraph to better understand the problem difficulty and the effectiveness of our method, as shown in the Fig. 3.

A.3 ABLATION STUDIES

We perform ablation studies for the GNN module, subtree module, and chordless cycle.

The GNN module in our analysis will capture the distributional features on the graph, such as the edge density difference between classes. The GNN module is thus essential for datasets with multiple distributions, also called biased data. We run experiments on both the biased and unbiased synthetic datasets to show the performance of our method and its variation that without the GNN module, as shown in Table 3. D^2Match outperforms D^2Match without the GNN module as our theory predicts. But our GNN module shares the same weaknesses as the other GNN models when dealing with evenly distributed data. We observe that the performance of the GNN module drops significantly on hard datasets similar to other GNN models. The subtree module can significantly improve the performance because it harnesses the property of subgraph-matched data, making it robust to data's distribution. Our subtree module outperformed the GNN module on all datasets in our ablation study, demonstrating its effectiveness.

Table 3: The ablation study of D2Match module

	Synthetic	Synthetic$^+$	Proteins	Proteins*	IMDB-Binary	IMDB-Binary*	FirstMMDB	FirstMMDB*
D^2Match (gnn only)	$61.1_{\pm 13.31}$	$70.2_{\pm 18.58}$	$95.2_{\pm 1.04}$	$77.2_{\pm 8.11}$	$50.0_{\pm 0.00}$	$64.4_{\pm 19.73}$	$69.7_{\pm 26.98}$	$67.8_{\pm 24.38}$
D^2Match (subtree only)	$70.0_{\pm 2.09}$	$74.8_{\pm 2.56}$	$100.0_{\pm 0.00}$	$82.0_{\pm 2.92}$	$92.9_{\pm 1.04}$	$82.8_{\pm 4.02}$	$100.0_{\pm 0.0}$	$72.0_{\pm 6.20}$
D^2Match	$72.7_{\pm 4.45}$	$86.6_{\pm 1.44}$	$100.0_{\pm 0.00}$	$83.4_{\pm 2.97}$	$93.3_{\pm 1.03}$	$90.2_{\pm 1.79}$	$100.0_{\pm 0.0}$	$86.4_{\pm 7.44}$

Table 7: The hard dataset details

	Synthetic$^+$	Proteins*	Mutag*	Enzymes*	Aids*	IMDB-Binary*	Cox2*	FirstMMDB*
Average nodes (target)	40.0	38.8	18.2	31.5	14.7	19.0	41.3	1376.7
Average nodes (query)	15.0	11.4	9.1	15.4	4.4	14.2	15.0	15.0
Average edges (target)	259.5	146.7	40.2	120.6	30.0	177.1	87.0	6141.6
Average edges (query)	67.3	35.5	17.6	52.6	7.1	102.6	29.9	45.6

We also perform the ablation study on the Synthetic dataset to test the effect of chordless cycles,as shown in Table 4. Results show the chordless cycles boost the performance with limited extra time consumption.

	Synthetic	RunTime
D^2Match	$74.3_{\pm 1.60}$	19.7s/epoch
D^2Match (w/o cc)	$72.7_{\pm 4.45}$	10.3s/epoch

Table 4: The ablation study of cc

	proteins	mutag
Seed(0)	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$
Seed(1)	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$
Seed(2)	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$
Fixed	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$

Table 5: Random seed comparison

	Training(s/epoch)	Inference(s/epoch)
SimGNN	1.732	0.385
NeuroMatch	2.234	0.311
GMN-embed	1.850	0.290
GraphSim	3.223	0.433
IsoNet	10.553	1.939
D^2Match-Subtree(S=2)	2.940	0.456
D^2Match-Subtree(S=3)	3.889	0.581
D^2Match-Subtree(S=4)	4.410	0.673
D^2Match-Subtree(S=5)	5.143	0.750
D^2Match-GNN	2.678	0.495
D^2Match	8.163	1.114

Table 6: Runtime analysis

A.4 Random Effect

Although our experiments do not rely on random seeds, a random split may affect the results. To test this, we set up several random seeds and permute the raw data order before getting the five-fold. We experiment on the Protein and Mutag datasets with trivial random seed 0,1,2 and obtain nearly identical performance. See Table 5.

While other methods based on GNNs tend to capture the divergence of distributions in the training set and hence are easily affected by randomness, our subtree module performs the matching explicitly by the degeneracy property, as opposed to modeling the data distribution in others, hence ours is insensitive to data partitioning.

A.5 Runtime Analysis

We add the runtime analysis experiment as follows. We compare our method with baselines on the synthetic dataset and record the training and inference time (second) per epoch. The results are shown in Table 6.

Our model is slower than some strong baselines like SimGNN and NeuroMatch in the experiment because they deal with the graph-level representations. Our model is faster than IsoNet, which performs edge-level matching.

We conduct an additional ablation study to explore the time consumption of each module in our model. The results show that the time consumption of our model mainly comes from the sampling in the subtree module whose running time is linearly related to the sampling number. When we set

Table 8: The dataset details

	Synthetic	Proteins	Mutag	Enzymes	Aids	IMDB-Binary	Cox2	FirstMMDB
Average nodes (target)	40.0	39.1	17.9	33.0	15.7	19.8	41.3	1376.5
Average nodes (query)	15.0	14.4	9.0	14.8	7.9	14.6	14.4	15.0
Average edges (target)	241.7	146.5	39.5	125.6	32.4	193.1	87.0	6144.3
Average edges (query)	50.6	68.9	25.1	75.3	17.1	141.0	42.8	68.1

the sampling number as 2, the running time is on par with the others. Furthermore, the running time for the GNN module is the same as for the other baselines. In sum, we observe that our model's scalability is acceptable as both complexity analysis and empirical running time show ours is slower than others only by a constant factor.

A.6 Dataset Details

We describe the average node number and average edge number of the target graph and query graph in the Table 8 and Table 7. Except the hard datasets, we generate 1000 graph pairs for Synthetic, Proteins, Mutag, Enzymes, Cox2 and FirstMMDB and 2000 graph pairs for Aids and IMDB-Binary which have smaller graph size. For the hard dataset, we uniformly generate 500 graph pairs.

A.7 Results on More Datasets

We conduct experiments on the OGB benchmark dataset (Hu et al., 2020), including Ogbg-molhiv and Ogbg-molpcb. We follow the same strategy in the paper to construct normal and hard versions for these datasets and choose the best-performing baselines for comparison, including SimGNN and NeuroMatch. We present new results in Table 9.

We find that our model performs slightly better than others on normal datasets while gaining a significant advantage over baselines on hard datasets. These results are consistent with our previous experiments, demonstrating that our model exploits the subgraph matching property, rather than simply modeling the divergence of the data distribution as other GNNs.

We experiment on continuous features from the MNIST, CIFAR10 and PPI datasets, as these are constructed from vision data(Dwivedi et al., 2020) or biological information data(Zitnik & Leskovec, 2017). We As expected, our model achieves consistent performance as well. See Table 10.

	ogb-molhiv	ogb-molhiv*	ogb-molpcba	ogb-molpcba*
SimGNN	$99.4_{\pm 0.65}$	$81.6_{\pm 2.70}$	$99.8_{\pm 0.27}$	$86.2_{\pm 2.28}$
NeuroMatch	$98.3_{\pm 1.68}$	$86.0_{\pm 3.54}$	$99.8_{\pm 0.27}$	$90.6_{\pm 3.51}$
D^2Match	$99.8_{\pm 0.27}$	$99.6_{\pm 0.54}$	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$

Table 9: Obg dataset performance comparison

	Cifar10	MNIST	PPI
SimGNN	$89.0_{\pm 21.82}$	$98.5_{\pm 0.93}$	$77.0_{\pm 24.67}$
NeuroMatch	$98.1_{\pm 1.14}$	$95.9_{\pm 1.34}$	$50.0_{\pm 0.00}$
D^2Match	$99.3_{\pm 0.27}$	$98.8_{\pm 1.15}$	$98.8_{\pm 1.06}$

Table 10: Continues dataset performance

A.8 Comparison with Exact Method

we compare exact matching solutions, including VF2[1] and ISMAGS[2]. By nature, we know that exact matching methods obtain 100 % accuracy.

As a trade-off between accuracy and execution time, we make the comparison inspired by the setup in NeuroMatch (Rex et al., 2020). We say an execution succeeds when its run time is less than 60s. We compare the success rate of the exact methods by varying the query graph size from 10 to 50 on the synthetic data, as shown in Figure.4.

We show in our experiment that the failure of exact matching methods increases significantly when the target graph has more than 30 nodes, compared to our stable performance, indicating the incompetence of these methods on large-scale datasets.

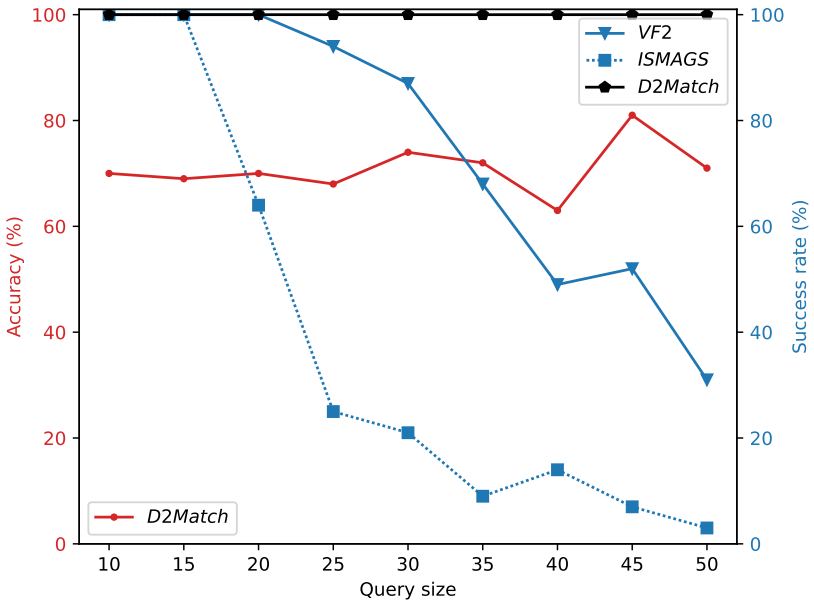


Figure 4: Comparison with exact method