A Supplementary Material on Methodology

A.1 Symbol Summary

Table 6 provides a summary of all symbols used in this paper, along with their descriptions.

Symbol	Description
$\mathcal{V}$	Node set
$\mathcal{E}$	Edge set
$\mathcal{X}$	Node features
A	Adjacency matrix
n	Number of nodes
m	Number of edges
$\mathcal{M}$	Target GNN model
Y	Graph/node classification label
E	Environment variables
G_c	Explanation subgraph
G_s	Complement graph after excluding the explanation subgraph
G_{test}	Graph in testing dataset
X_{str}	Structure-based features for nodes
H_{str}	Structure-based embeddings for nodes
H_G	Structure-based embeddings for graphs
K	Number of potential environments
env	Environmental embedding set
E_{str}	Structure-based environment label set
E_k^s	The k-th structure-based environment label
E_{feat}	Feature-based environment label set
E_k^f	The k-th feature-based environment label
$\mathcal{V}_c$	Nodes causally related to the classification label
$\mathcal{V}_s$	Nodes causally related to the environment
$\mathbf{H}$	Feature-based node embeddings
$\mathbf{h}_i$	Feature-based embedding for node i
$\mathbf{h}_G$	Feature-based embedding for graph G
$\mathbf{e}_i$	Environmental embedding for node i
$\mathbf{e}_G$	Environmental embedding for graph G
$\boldsymbol{\mu}_i$	Mean of the node-invariant representation distribution of node i
$\boldsymbol{\mu}_G$	Mean of the graph-invariant representation distribution of G
$\log(\boldsymbol{\sigma}_i^2)$	Log-variance of the node-invariant representation distribution of node i
$\log(\boldsymbol{\sigma}_G^2)$	Log-variance of the graph-invariant representation distribution of graph G
$\mathbf{z}_i$	Node-invariant representation of node i
$\mathbf{z}_{\text{node}}$	Node-invariant embeddings
$\mathbf{z}_G$	Graph-invariant embedding of graph G
$\boldsymbol{\epsilon}$	Random noise
$D_{\text{KL}}(\cdot \Vert \cdot)$	KL divergence
$JS(\cdot \Vert \cdot)$	JS divergence
$q_{\phi_1}(\mathbf{z}_i \mid \mathbf{h}_i, env)$	Distribution modeled by the NodeVAE encoder
$p_{\theta_1}(\mathbf{h}_i \mid \mathbf{z}_i, env)$	Distribution modeled by the NodeVAE decoder
$q_{\phi_2}(\mathbf{z}_G \mid G, env)$	Distribution modeled by the GVAG encoder
$p_{\theta_2}(v_i \mid \mathbf{z}_G, \mathbf{z}_i, env)$	Node existence probability distribution modeled by GVAG decoder
$p_{\theta_3}(e_{ij} \mid \mathbf{z}_G, \mathbf{z}_i, \mathbf{z}_j, env)$	Edge existence probability distribution modeled by GVAG decoder
$p(\mathbf{z})$	Prior distribution of graph-invariant representations
$\text{Prob}(v_i)$	Node existence probability of node i
$\text{Prob}(e_{ij})$	Edge existence probability of edge e_{ij}
$\mathcal{L}_{\text{MI}}$	Subgraph reconstruction loss/MI loss
$\mathcal{L}_{\text{RR}}$	Reconstruction regularization loss
$\mathcal{L}_{\text{NodeVAE}}$	NodeVAE loss
$\mathcal{R}_{\text{causal}}$	Causal structure regularization
$\mathcal{R}_{\text{hinge}}$	Hinge regularization
$\mathcal{R}_{\text{subg\_node}}$	Subgraph node count regularization
$\mathcal{L}_{\text{CON}}$	Contrastive loss
LAR	Last action rewards
fid_-	Negative Fidelity
fid_+	Positive Fidelity
GEF	Unfaithfulness
ρ_v	Node density relative to the original graph
ρ_e	Edge density relative to the original graph
T	Response time (second)

Table 6: List of symbols and their descriptions.

A.2 Subgraph Reconstruction Algorithm

Algorithm 1 presents the pseudocode for generating subgraphs during the training phase. In essence, during the training phase, we first sample nodes that will appear on the explanation subgraph based on their existence probability, assigning these sampled nodes a probability of 1.0. Then, we recalculate the existence probabilities for each edge by integrating

Algorithm 1 Sample-based subgraph reconstruction algorithm

Require: Input parameters:
edge_index: Edge set of original graph
node_prob: Node existence probability
link_prob: Edge existence probability
max_nodes: Maximum number of nodes in subgraph
start_nid: Start node of generating subgraph
density: Subgraph density limit
max_iter: Maximum number of iterations
min_edges: Minimum number of edges in subgraph

Initialize:
Ensure *max_nodes* does not exceed the number of nodes available in the original graph
Adjust *min_edges* based on the *density*
Add a small epsilon to *node_prob* and *link_prob* to prevent computational issues
Initialize node selection vector *current_node*
Sampling nodes for the subgraph:
if *start_nid* is provided **then**
    Set the corresponding index in *current_node* to True
end if
Copy *node_prob* as sampling probabilities $Prob_n$
for *iter* from 1 to *max_iter* **do**
    For nodes in *current_node*, set $Prob_n$ to -1
    Sample new nodes based on $Prob_n$ and update *current_node*
    if sum(*current_node*) $\geq$ *max_nodes* **then**
        break
    end if
end for
if sum(*current_node*) $<$ *max_nodes* **then**
    For nodes in *current_node*, set $Prob_n$ to -1
    Select new nodes based on $Prob_n$ and update *current_node*
else
    Prune nodes from *current_node* based on *node_prob* to fit within *max_nodes*
end if
Reset $Prob_n$ to *node_prob*
For nodes in *current_node*, set $Prob_n$ to 1
Sampling edges for the subgraph based on the nodes within the subgraph:
Initialize edge selection vector *current_edge*
for *iter* from 1 to *max_iter* **do**
    Copy *link_prob* as sampling probabilities $Prob_e$
    For links in *current_edge*, set $Prob_e$ to 0
    Recompute $Prob_e$ using $Prob_n$ and $Prob_e$
    Sample links based on $Prob_e$
    Update *current_link* based on sampled links
    if edge density $>$ *density* **then**
        break
    end if
end for
If node has no edge, remove it from *current_node*
Calculate total graph probability *total_graph_prob* based on *node_prob* and *link_prob*

return *current_node, current_link, total_graph_prob*

Algorithm 2 Edge first reconstruction algorithm

Require: Input parameters:
edge_index: Edge set of original graph
node_prob: Node existence probability
link_prob: Edge existence probability
max_nodes: Maximum number of nodes in subgraph
start_nid: Start node of generating subgraph
density: Subgraph density limit
min_edges: Minimum number of edges in subgraph

Initialize:
Initialize node selection vector *current_node*
Initialize edge selection vector *current_edge*
Adjust *min_edges* based on *density*
Add a small epsilon to *node_prob* and *link_prob* to prevent computational issues

Prepare edge existence probability:
if *start_nid* is provided **then**
    Set the corresponding index in *current_node* to True
end if
total_graph_prob ← *0.0*
current_node_prob ← *node_prob*
current_link_prob ← *link_prob*
Recompute *current_link_prob* using *current_node_prob* and *current_link_prob*
max_edges ← *ceil(density* * $|edge\_index|$*)*
if *max_edges* ≤ *min_edges* **then**
    max_edges ← *min_edges*
end if

Select edges for the explanatory subgraph based on their probabilities:
sorted_edge_prob, sorted_eid ← *topk(current_link_prob, k=max_edges)*
For edge id in *sorted_eid*, set *current_edge* to True
Calculate total graph probability *total_graph_prob* based on *sorted_edge_prob*

Update Nodes Based on Selected Edges:
Get source nodes of edges in *current_edge*, as *src_nodes*
Get destination nodes of edges in *current_edge*, as *dst_nodes*
For nodes in *src_nodes* or *dst_nodes*, set *current_node* to True

return *current_node, current_link, total_graph_prob*

both the node existence probability and the edge existence probability relevant to the current subgraph. Next, these recalculated probabilities are subsequently utilized to sample edges that will appear on the explanation subgraph. This methodology ensures that the probabilities of both nodes and edges are considered concurrently in generating the explanation subgraph, which can help in maintaining connectivity within the subgraph.

Moreover, through this random sampling process, GVAG effectively explores the entire space of potential subgraphs and avoids focusing on a limited set of nodes or edges, thereby enhancing the robustness and generalization of the generated explanation.

In the testing phase, GVAG directly uses the node existence probability and edge existence probability to calculate the final probability of each edge appearing on the explanation subgraph, and add these edges to the explanation subgraph according to the probability. The corresponding pseudocode is presented in Algorithm 2.

A.3 Computational Complexity Analysis of Subgraph Generation During Training Phase

We adopt a sampling-based subgraph generation algorithm, detailed in Algorithm 1, with its complexity influenced by various operations. The initialization steps, including adjustments to `max_nodes` and `min_edges`, are constant operations with a complexity of $O(1)$, while adding ϵ to zero probabilities involves linear operations, resulting in a combined complexity of $O(n) + O(m)$. The node sampling loop iterates up to `max_iter` times, with each iteration involving probability calculations and updates for all nodes, leading to a complexity of $O(n \cdot \text{max\_iter})$. Similarly, the edge sampling loop processes all edges within each iteration, requiring recomputation of probabilities and sampling, contributing a complexity of $O(m \cdot \text{max\_iter})$. Post-processing adjustments, such as sorting and selecting top k elements for nodes and edges, add logarithmic factors, typically $O(n \log n)$ and $O(m \log m)$, respectively. Considering $n > m$, $\text{max\_iter} > \log n$, and $\text{max\_iter} > \log m$, the overall time complexity of the algorithm is $O(n \cdot \text{max\_iter})$.

A.4 Computation Complexity Analysis of Subgraph Generation During Evaluation Phase

The computational complexity of the Algorithm 2 primarily depends on the operations performed on nodes and edges within the graph. Initially, adjusting the probability vectors for zero probabilities, which are operations linear in terms of the number of nodes n and edges m, contributes a complexity of $O(n + m)$. This setup is followed by the critical step of selecting edges based on updated probabilities, involving a sorting operation. Since the edges are sorted to select the top edges based on their probability, this step incurs a complexity of $O(m \log m)$, which is the most computationally intensive part of the function. Updating the node selection based on the edges selected is relatively straightforward and operates linearly with respect to the number of selected edges, hence contributing an additional linear term. Overall, the sorting of edge probabilities dominates the computational complexity, making the function's total complexity mainly governed by $O(m \log m)$ with an additional linear component due to initialization and node updates based on selected edges.

B Supplementary Material on Experiments

B.1 Experimental Setup

Our experimental were conducted on an AMD EPYC 9754 CPU, an NVIDIA 4090D GPU with 24GB G6X memory, and 60GB of RAM. The software environment includes Python 3.10, CUDA 12.1, and PyTorch 2.1.0.

The Hyper-Parameter Settings. The key hyper-parameter settings for training and evaluation are shown in Table 7. These hyper-parameters are carefully selected based on prior empirical results and tuning experiments to balance model fidelity and efficiency.

B.2 Analysis of Hyper-Parameter Sensitivity

We also test the impact of several hyper-parameters on the quality of the final generated explanation subgraph, including

Hyper-parameters	Cora	Motif
Learning Rate	0.01	0.005
Weight Decay	1.00E-04	1.00E-04
Structure Infer Epochs	5	5
Number Environments K	4	5
Number Epochs	1	10
Batch Size	64	64
Prior Subgraph Max Nodes	60	7
Prior Subgraph Min Nodes	15	5
Prior Subgraph Density	0.35	0.1
Recon Loss Weight ω_{RECON}	2	2
Contrastive Loss Weight ω_{CON}	0.5	0.5
Last Action Rewards Weight ω_{LAR}	1	1

Table 7: Hyper-parameters settings.

edge density, last action rewards weight and reconstruction weight.

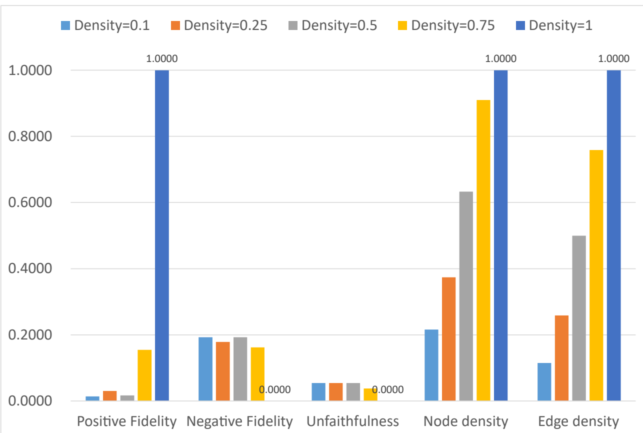

Figure 3: Hyper-parameter sensitivity study on different edge densities.

Different Edge Density. This study examines how edge density impacts the model's ability to generate effective explanations, with results summarized in Figure 3. As edge density increases from 0.1 to 1.0, a distinct trend in the performance metrics emerges:

- **Positive Fidelity:** Variations in positive fidelity suggest that explanation subgraphs with higher densities include more critical features essential for GNN predictions, thereby enhancing positive fidelity by better aligning with GNN predictions.
- **Negative Fidelity:** Adjustments in negative fidelity with varying densities indicate that lower densities, which result in sparser subgraphs, may omit crucial features necessary for the GNN's decision-making process.
- **Unfaithfulness:** Changes in unfaithfulness show that denser explanation subgraphs are likely to exhibit lower unfaithfulness, implying that denser subgraphs may better align with the predictions of the original graph, thus offering more faithful interpretations.

This analysis underscores the importance of managing edge density to balance the trade-offs between comprehensiveness and simplicity in explanation subgraphs. Lower densities, while easier to interpret, might miss critical information; conversely, higher densities, though potentially more complex, provide a more detailed and accurate representation of the factors influencing the model's decisions. Achieving this balance is crucial for ensuring that explanations are both informative and practically useful, therefore meeting the needs of real-world applications.

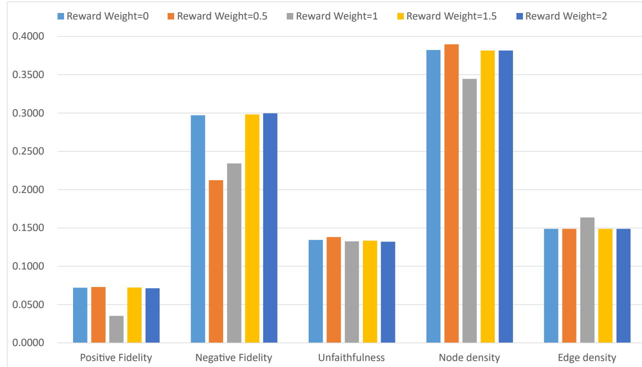

Figure 4: Hyper-parameter sensitivity study on different weights of LAR.

Different Weights of Last Action Rewards. This study explores how varying the weights of the last action rewards influences OPEN's performance, with results presented in Figure 4.

- **Positive Fidelity:** Positive fidelity remains relatively stable across different reward weights, suggesting that adjustments in the last action rewards do not significantly affect the model's capability to include critical graph structures in the explanation subgraphs.
- **Negative Fidelity:** We observe a regular increase in negative fidelity as the reward weight increases, indicating that excessively high reward weights might overly penalize the model's errors, potentially leading to the exclusion of relevant structures.
- **Unfaithfulness:** Unfaithfulness demonstrates a decreasing trend with higher reward weights, which implies that the explanations become more aligned with the original graph's predictions, enhancing their faithfulness and reliability.

These findings indicate that while a higher reward weight can enhance the faithfulness of explanations, it may simultaneously compromise negative fidelity by penalizing the model too harshly. Therefore, it is essential to finely tune the last action rewards to maintain a balance between the depth and accuracy of the explanations produced by OPEN, ensuring that they are comprehensive yet precise.

Different Weights of Reconstruction Loss. This study examines the effect of varying reconstruction loss weights on OPEN's performance, with results illustrated in Figure 5.

- **Positive Fidelity:** Positive fidelity shows a positive correlation with increasing reconstruction loss weight, climbing from 0 to 2. This trend suggests that higher weights prompt

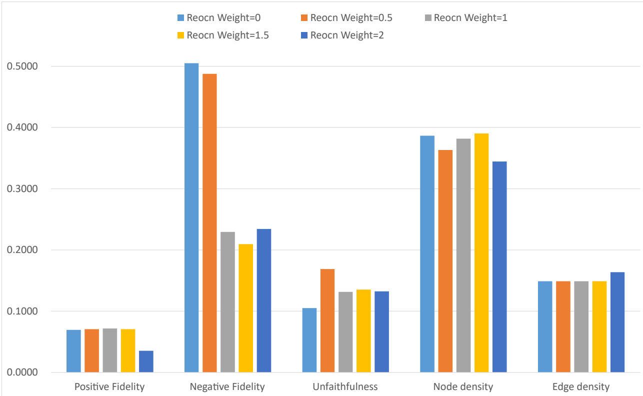

Figure 5: Hyper-parameter sensitivity study on different weights of reconstruction loss.

the model to preserve more crucial structures within the explanation subgraphs, thereby boosting positive fidelity.

- **Negative Fidelity:** Conversely, negative fidelity decreases as the reconstruction loss weight increases, highlighting that enhanced penalties for incorrect explanations help the model omit non-essential structures, thus refining the fidelity of its explanations.
- **Unfaithfulness:** Unfaithfulness does not exhibit a consistent trend with changes in reconstruction loss weight, indicating that while reconstruction loss aids in refining explanations, its effect on the faithfulness may be influenced by other factors within the model.

These findings illustrate the critical role that reconstruction loss weight plays in explanations generated by OPEN. Properly setting this weight can enhances the fidelity and accuracy of explanations.

B.3 Explanation Subgraph Examples

Figure 6 displays explanation subgraphs generated by the proposed OPEN and some other XGNN methods on the Motif dataset in the basis domain. The figure reveals that all methods, except PGMExplainer, successfully identify nodes relevant to predictions (blue nodes). OPEN stands out by considering both node and edge existence probabilities during subgraph generation, ensuring the connectivity of the explanation subgraphs. Moreover, OPEN's ability to adjust the size and density of the subgraphs based on prior knowledge offers a more flexible and user-friendly explanation approach compared to existing XGNN methods, enhancing user comprehension.

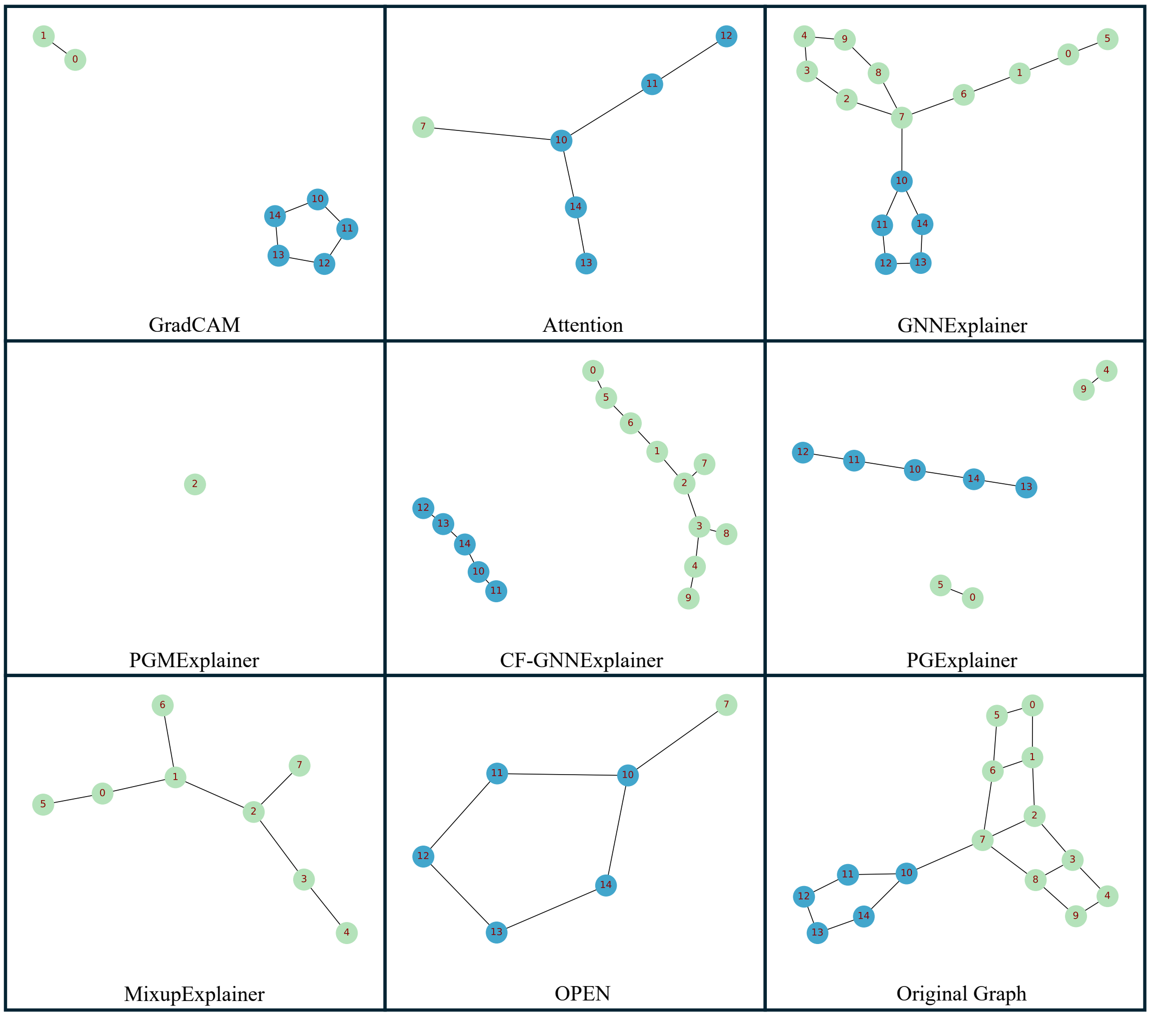

Figure 6: Explanation subgraphs.