# Adaptive Subgraph Neural Network with Reinforced Critical Structure Mining

Jianxin Li, *Member, IEEE*, Qingyun Sun, Hao Peng, *Member, IEEE*, Beining Yang, Jia Wu, *Senior Member, IEEE*, and Phillp S. Yu, *Life Fellow, IEEE*

**Abstract**—While graph representation learning methods have shown success in various graph mining tasks, what knowledge is exploited for predictions is less discussed. This paper proposes a novel **A**daptive **S**ubgraph **N**eural **N**etwork named **AdaSNN** to find critical structures in graph data, i.e., subgraphs that are dominant to the prediction results. To detect critical subgraphs of arbitrary size and shape in the absence of explicit subgraph-level annotations, AdaSNN designs a Reinforced Subgraph Detection Module to search subgraphs adaptively without heuristic assumptions or predefined rules. To encourage the subgraph to be predictive at the global scale, we design a Bi-Level Mutual Information Enhancement Mechanism including both global-aware and label-aware mutual information maximization to further enhance the subgraph representations in the perspective of information theory. By mining critical subgraphs that reflect the intrinsic property of a graph, AdaSNN can provide sufficient interpretability to the learned results. Comprehensive experimental results on seven typical graph datasets demonstrate that AdaSNN has a significant and consistent performance improvement and provides insightful results.

**Index Terms**—Graph representation learning, graph neural network, graph classification, critical structure, reinforcement learning, mutual information.

## 1 Introduction

Graph is widely-used to represent and organize data with complicated relationships [1], [2]. Analyzing the underlying properties of data with graph structure is fundamental and important in the graph mining area. However, it is hard to process graph data by machine learning methods directly due to its non-Euclidean property, i.e., arbitrary size and complex topological structure [3], [4], [5]. Over the last decade, lots of efforts have been made to extend deep learning on graphs [6] in a growing number of real-world applications including anomaly detection [7], [8], fraud detection [9], drug discovery [10], community analysis [11], link prediction [12], etc. Graph representation learning [13] is one of the key techniques, which aim to extract underlying information of the graph-structured data by projecting it into low dimensional vector representations.

Graphs have a wide spectrum of local structures, ranging from nodes, and motifs, to subgraphs. Prevailing works show that the vital characteristics and prominent patterns of a graph are reflected through local structures such as motifs and subgraphs [14], [15]. Many research efforts have been focused on preserving graph topology information, ranging from shallow kernel-based methods to deep graph neural networks. Graph kernel methods [16] decompose graph substructures by kernel functions and directly exploit them to measure the similarity of pairs of graphs rather than vectorization. Graph neural networks (GNNs) [17], [18], [19], [20] generally follow a message-passing scheme, which aggregate neighbor information recursively between a neighborhood subgraph of a central node. Even though GNNs show remarkable success in many graph mining tasks, most of them only propagate information across edges for node representations and globally summarize them for an overarching graph representation, which is inherently flat. The global memorization includes averaging all node representations [21], adding virtual nodes [5], using fully-connected layers [17], [22] or convolutional layers [23], etc. However, these flat GNNs share a drawback in that they detect only fuzzy patterns and are unable to capture precise distinguishable structural information effectively. Recently, researchers attempt to extract the hierarchical substructures of graphs via hierarchical graph pooling aggregation [24], which greatly improves the ability of graph representation.

It's found that the properties of a graph are mainly determined by some critical structures [25], [26], [27]. For example, molecular toxicity is determined by several functional groups. Classmate relationships are important for interest recommendation but not for hometown classification. Investigating critical structures (i.e., structures that are dominant to a prediction in the task) is crucial toward pushing forward the performance and explanation boundary of GNNs. Most of the current works exploit the critical structure pattern based on heuristic intuitions or experimental trial-and-errors, and the critical structures of very simple shapes such as ego-networks [28] or motifs [29], [30]. However, the following challenges make the critical structure detection task nontrivial: (1) **Data-Specific and task-specific.** Critical structures can be different across different datasets and tasks. It's hard to predefine their patterns. (2) **Arbitrary size and shape.** Critical structures can be larger structures

- *J. Li, Q. Sun, H. Peng, and B. Yang are with Beijing Advanced Innovation Center for Big Data and Brain Computing, School of Computer Science and Engineering, Beihang University, Beijing 100083, China. E-mail: {lijx, sunqy, penghao, yangbn}@act.buaa.edu.cn;*
- *J. Wu is with with Department of Computing, Macquarie University, Sydney, NSW 2109, Australia. E-mail: jia.wu@mq.edu.au;*
- *PS. Yu is with the Department of Computer Science, University of Illinois at Chicago, Chicago 60607, USA. E-mail: psyu@uic.edu.*

*Manuscript received April 22, 2022.*

of varying sizes and shapes. It's hard to design fixed rules as extract constraints. (3) **Absence of annotations.** For most datasets and application scenarios, the annotations of critical structures are absent. The absence of annotations makes it difficult to utilize the valuable information of critical structure in the learning process in the form of supervision information. It's hard to mine critical structures in an unsupervised way. Even though we have some expert knowledge, this knowledge cannot be transferred to other tasks. (4) **Local-Scale and global-scale.** Critical structures should not only contain local semantics (at local-scale) but also play important roles in the global graph properties (at global-scale).

**Present work.** This paper proposes **AdaSNN**[1], a novel **Ada**ptive **S**ubgraph **N**eural **N**etwork for graph classification, to tackle all of the challenges above. AdaSNN's core principle is to detect critical subgraphs as a representative abstraction of the graph, which can navigate the complexity of customized hierarchical structures whilst maintaining discriminative graph representations. AdaSNN reveals critical subgraph-level patterns in a sketched graph by three steps: *(1) critical subgraph detection*, *(2) subgraph sketching and encoding*, and *(3) subgraph representation enhancement*. The hierarchical structure contraction preserves structure properties hierarchically: node, intra-subgraph, and inter-subgraph. We aim to provide an effective and adaptive framework for critical subgraph detection without domain knowledge and learn discriminative representations with good generalization performance in an explanatory way.

A preliminary version of this work appeared in the proceedings of the Web Conference 2021 [31]. This journal version has extended the preliminary two phase "sampling-selection" critical subgraph detection model SUGAR to an adaptive critical subgraph exploration architecture. This journal version involves several improvements in upgrading the framework of the subgraph neural network with significant aspects. First, we designed a novel **Reinforcement Subgraph Detection Module** to detect critical subgraphs. It can find critical subgraphs adaptively according to the feedback of downstream tasks, solving the annotation absence problem effectively. Compared with SUGAR, the critical structures detected by AdaSNN are with varied sizes and shapes, which are not constrained to inflexible sampling rules and needless domain knowledge. Second, we design a **Bi-Level Mutual Information Enhancement Mechanism** including both global-aware MI and label-ware MI, which encourages the detected subgraphs not only to be aware of global graph properties but also to be predictive for the downstream task. Third, we carry out extensive experiments compared with more competitive baselines on a large-scale bioinformatics dataset and three representative social network datasets. We also evaluate the robustness of AdaSNN on the graph denoising task. We highlight the advantage of AdaSNN as follows:

- **Multi-Resolution.** AdaSNN starts with node-level attribute embeddings and continues with subgraph-level pattern combination, and all the way up to graph-level representation, providing a multi-resolution framework for graph representation learning.
- **Adaptive.** AdaSNN is a data-driven and task-driven framework trained under the supervision of ground-truth graph labels. The integration of reinforcement learning and graph learning makes AdaSNN perform supremely across various graph data domains.
- **Interpretable.** AdaSNN directly reveals the vital and label-relevant subgraphs dominating the learned result rather than learning a representation of the input data in the hidden space, which provides intuitive and insightful interpretation into downstream applications.

The remainder of this paper is organized as follows. Section 2 introduces the background and related work. Sections 3 proposes the overall architecture of AdaSNN and then introduces the Reinforced Subgraph Detection Module and the Bi-Level MI maximization mechanism, respectively. In Section 4, we describe the experimental methodology and present the results. Finally, Section 5 concludes this work and suggests future research directions.

## 2 Related Work

In this section, we will discuss the following two lines of related work including graph neural networks, critical structure mining in graph representation learning, GNNs with reinforcement learning, and the GNNs with mutual information mechanism.

### 2.1 Graph Neural Networks

Graph Neural Networks (GNNs) [24] extend the traditional convolution operator in regular domains to arbitrary and unordered graph-structure data, including spatial-based [17], [32], [33] and spectral-based methods [34], [35], [36], [37]. Generally, GNNs project graph data into a continuous and low-dimensional space by a message-passing scheme recursively [17]. Most of them globally fuse node features to generate overarching graph representations [32], [38], [39], [40], [41], which fail to capture the hierarchical structure properties.

Some graph pooling methods [23], [42], [43], [44], [45] try to propose some principles to extract structure information hierarchically, which can be broadly divided into two categories: cluster based pooling and selection based pooling. *Cluster based pooling methods* cluster nodes into groups and then generate a coarsened graph based on the group assignments. Typical methods include DiffPool [42], EigenPool [46] and ASAPool [45], etc. This kind of method utilizes feature and structure information by clustering implicitly, which leads to a lack of interpretability. *Selection based pooling methods* [23], [44], [47] select important nodes by calculating their importance scores to remove redundant information. SortPool [23] learns nodes' structure roles using the WL kernel and sorts nodes according to that. Top-k Pool [44] learns the nodes' importance values by a trainable vector and selects the most important K of them. SAGPool [43] directly trains a binary classifier to decide the preserving nodes. Most selection pooling methods apply the node-level strategy, ignoring the semantic information of local substructures.

[1] The code of this work is publicly available at https://github.com/RingBDStack/AdaSNN.

TABLE 1: Comparison of graph representation methods in the perspective of critical structure mining.

| Graph embedding model | Data-specific and task-specific | Arbitrary size and shape | Local Scale | Global Scale | Without fixed rules or prior knowledge |
|---|---|---|---|---|---|
| Graph Kernel ( [48], [49]) | | | ✓ | | |
| Spectrum GNNs ( [35], [50], [51]) | ✓ | | | | |
| Spatial GNNs ( [18], [19], [52]) | ✓ | | ✓ | | |
| Clustering Pooling ( [42], [45], [46]) | ✓ | | ✓ | | |
| Selection Pooling ( [23], [43], [44]) | ✓ | | ✓ | | |
| Motif-based GNNs ( [30], [53]) | | | ✓ | | |
| Graph transformers ( [54], [55], [56]) | ✓ | | ✓ | | |
| Ego-CNN [28] | ✓ | | | ✓ | ✓ |
| NCAT [26] | ✓ | | ✓ | | ✓ |
| SIB [27], [57] | ✓ | ✓ | | ✓ | ✓ |
| SUGAR [31] | ✓ | | | ✓ | ✓ |
| AdaSNN(Ours) | ✓ | ✓ | ✓ | ✓ | ✓ |

To sum up, despite the popularity and importance of structures for graph representation learning [53], [58], there is still limited research on critical structure discovery.

### 2.2 Critical Structure Mining

Local substructures contain more complex structural characteristics and some of them play key roles that determined the model's prediction. Many methods have been proposed to mine the critical structure.

Graph kernel [16] is a conventional way for critical structure extraction, which directly use kernel functions to decompose pre-defined structures (e.g., walk [59], shortest path [60], subgraph [48]) from the original graph structure. Due to their specialization, graph kernels show competitive performances in some particular domains. However, the critical structure patterns in form of kernel function are most handcrafted and heuristic, which suffers from poor flexibility and generalization performance.

The neighborhood subgraphs are widely-used as critical structures. Ego-CNN [28] stacks up multiple ego-convolution layers to detect precise critical structures, but it can only detect the local ego-networks with $K$ nearest neighbors.

Motifs (i.e., frequently-occurring subgraph patterns) are also widely used to preserve local structure properties [29], [30], [61]. NEST [53] combine motifs to explores various subgraph-level patterns. Motif-based methods are limited to enumerating small local substructures as local features by predefined motif extraction rules, which require to be manually designed with domain expert knowledge.

Graph transformers [54], [55], [56] have shown outstanding performance in recent works. Rather than using the structure explicitly as GNNs, most of existing graph transformers [54], [55] capture interaction information between node pairs by the self-attention mechanism. As a result, they only encode positional relationships between nodes and fail to identify structural similarities between nodes, and can only identify the critical edges. SAT [56] uses the k-hop subgraphs or k-subtrees to extract structure information. However, the critical structure found by SAT is still constrained by the fixed construction rules.

Find more flexible subgraphs is also attract some research attention [25], [26], [27]. SGN [25] predefines detection rules to select appropriate subgraphs, expanding the structural feature space effectively. Compared to the proposed AdaSNN, SGN detects and selects subgraphs by heuristic rules, and the provided information is insufficient when subgraphs are large. NCAT [26] matches fixed-size combinatorial neighbors with learnable patterns and assigns different weights to each combination. SIB [27], [57] learns an assignment matrix for each edge directly to recognize the information bottleneck subgraph, which ignores the semantics of local subgraphs. CAL [62] discovers the critical subgraphs by estimating the causal and shortcut features via an attention module.

To sum up, the detection of data-specific and task-specific critical structures at both local scale and global scale without fixed rules or prior knowledge remains an important but unsolved problem. Table 1 compares our AdaSNN with existing graph representation learning methods in the perspective of critical structure mining. To the best of our knowledge, AdaSNN is the first method that can detect critical subgraphs of arbitrary size and shape in both local-scale and global-scale adaptively without prior knowledge of the dataset.

### 2.3 GNNs with Reinforcement Learning

Reinforcement learning (RL) [63] is one of the basic machine learning paradigms, which learns from the reward of the environment rather than labeled data. Recently, RL is incorporated into many graph mining tasks [64] including graph representation learning [65], graph adversarial attack [66], relational reasoning [67], GNN explainer [68] and combination optimization [69]. Similar to our AdaSNN, some methods employ RL to find critical structure to boost the ability of graph learning. Most of them (e.g., CARE-GNN [70], Rio-GNN [71], GDPNet [72] and FinEvent [73]) use RL to select critical neighbors to improve the effectiveness of GNNs. Policy-GNN [65] uses the DQN [74] to select the optimal iterations of aggregations for different nodes. RL-HGNN [75] adaptively designs meta-paths for the heterogeneous graph by a policy network. GraphNAS [76] designs an RL algorithm to search the optimal architectures of graph neural networks.

To sum up, current GNNs with RL focus on finding critical relations or better GNN architectures for node-level tasks. There is still limited research on advancing RL to find more complicated critical structures for graph-level tasks.

### 2.4 GNNs with Mutual Information

Mutual Information (MI) [77] provides a unified way to measure the degree of correlation between two variables, which is widely used in many applications. Recent works [78], [79], [80] leverage MI for self-supervised graph learning. DGI [81] learns node representations by maximizing MI between the patch representations and the summaries of graphs. InfoGraph [78] maximizes the MI between the graph representation and the representations of substructures to encode structure across different scales. GMI [79] generalizes the computation of MI from vector space to the graph space in terms of features and topology for node representation learning.

To sum up, most existing GNNs with MI focus on node-level or graph-level representations while ours focuses on subgraph representations. Besides, most of them exploit MI for self-supervised learning and seldom consider the MI between representations and labels, while our label-aware MI enhancement mechanism aims to make the subgraph more predictive for the downstream tasks.

## 3 AdaSNN: Adaptive Subgraph Neural Network

AdaSNN is an adaptive subgraph neural network for graph classification, which detects critical subgraphs as a representative abstraction of the whole graph. Concretely, a flexible architecture is required to *(1) detect critical subgraph of arbitrary size and shape without annotation supervision information* and *(2) preserve the properties of the whole graph in the detected critical subgraphs*. To solve these two problems, we design a Reinforcement Subgraph Detection Module and a Bi-Level MI Enhancement Mechanism, respectively.

In this section, we first give the problem formulation in Section 3.1 and present AdaSNN's pipeline in Section 3.2. Then we introduce the Reinforcement Subgraph Detection Module in Section 3.3, following the Bi-Level MI Enhancement Mechanism in Section 3.4. Finally, Section 3.5 describes the overall algorithm and optimization.

### 3.1 Problem Formulation

In this subsection, we first summarize the main notations in Table 2 and then formalize the definition of graph classification and critical subgraph detection as follows.

A set of graphs is denoted by $\mathbf{G} := \{G\}$. Each $G = (V, X, E, A)$ is represented by the set of nodes $V = \{v_i\}|_{i=1}^{|V|}$, node feature matrix $X \in \mathbb{R}^{|V|*d_0}$, the set of edges $E = \{e_i\}|_{i=1}^{|E|}$, and the adjacency matrix $A \in \mathbb{R}^{|V|*|V|}$, where $|V|$ and $|E|$ denote the sizes of $V$ and $E$, and $d_0$ denotes the dimension of node features. In this paper, we focus on the graph supervised classification task.

**Definition 1** (Graph Classification)**.** *Given a set of graphs* $\mathbf{G} := \{G\}$*, a graph classification algorithm learns a model* $\mathcal{F} : G \to Y$ *to map the graph sample to its label.*

**Definition 2** (Critical Subgraph Detection)**.** *Given a graph* $G$ *with label* $Y$*, a critical subgraph detection algorithm learns a model* $\mathcal{F} : G \to \{g\}$ *to find critical subgraphs* $\{g\}$ *that dominate the properties of graph* $G$ *with respect to its label* $Y$ *of the downstream task.*

TABLE 2: Main notations in the paper. The top rows are for graph representation learning; the middle rows cover deep reinforcement learning; the bottom rows cover mutual information maximization.

| *Notation* | Definition |
|---|---|
| $\mathbf{G}$ | A graph dataset. |
| $G, Y$ | A graph and its label. |
| $V, X$ | Node set and node attribute matrix. |
| $E, A$ | Edge set and adjacent matrix. |
| $G^{ske}$ | Sketched graph. |
| $V^{ske}, E^{ske}$ | Node set and edge set of the sketched graph. |
| $v$ | node. |
| $\mathcal{N}^{(k)}(v)$ | $k$-hop neighborhood of the node $v$. |
| $\mathbf{h}^g$ | Embedding of node in subgraph $g$. |
| $g$ | subgraph. |
| $V(g)$ | Node set of subgraph $g$. |
| $\mathbf{z}^g$ | Subgraph embedding. |
| $\mathbf{z}^G$ | Graph embedding. |
| $N$ | Number of detected subgraphs. |
| $b_{com}$ | The edge threshold of the sketched graph. |
| $\mathcal{L}_{cls}$ | Graph classification loss. |
| $\mathcal{S}$ | State set. |
| $\mathcal{A}$ | Action set. |
| $\mathcal{R}$ | Reward function. |
| $\mathcal{T}$ | State transition. |
| $s_t$ | State in the $t$-th timestep. |
| $a_t$ | Action in the $t$-th timestep. |
| $r_t$ | Reward in the $t$-th timestep. |
| $Q(s, a)$ | Value function to evaluate the state and action. |
| $\pi$ | Policy function. |
| $K$ | Maximum subgraph depth. |
| $p(\cdot)$ | Probability desity function. |
| $H(\cdot)$ | Shannon entropy. |
| $\mathcal{I}(\cdot, \cdot)$ | Shannon mutual information. |
| $\mathrm{KL}(\cdot\|\|\cdot)$ | Kullback Leiber divergence. |
| $n^{neg}$ | The number of negative samples. |
| $\mathcal{L}_{MI}^{global}$ | Global-Aware MI enhancement loss. |
| $\mathcal{L}_{MI}^{label}$ | Label-Aware MI enhancement loss. |

### 3.2 Pipeline of AdaSNN

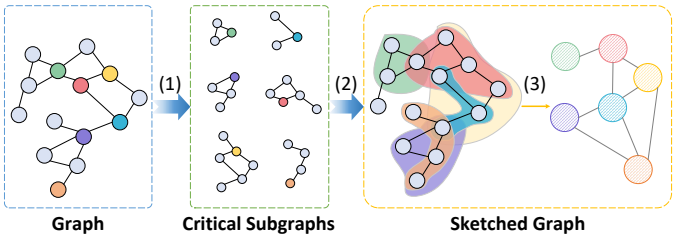


Fig. 1: The pipeline of AdaSNN includes three steps to build a subgraph neural network: (1) Critical Subgraph Detection, (2) Subgraph Sketching and Encoding, and (3) Subgraph Representation Enhancement.

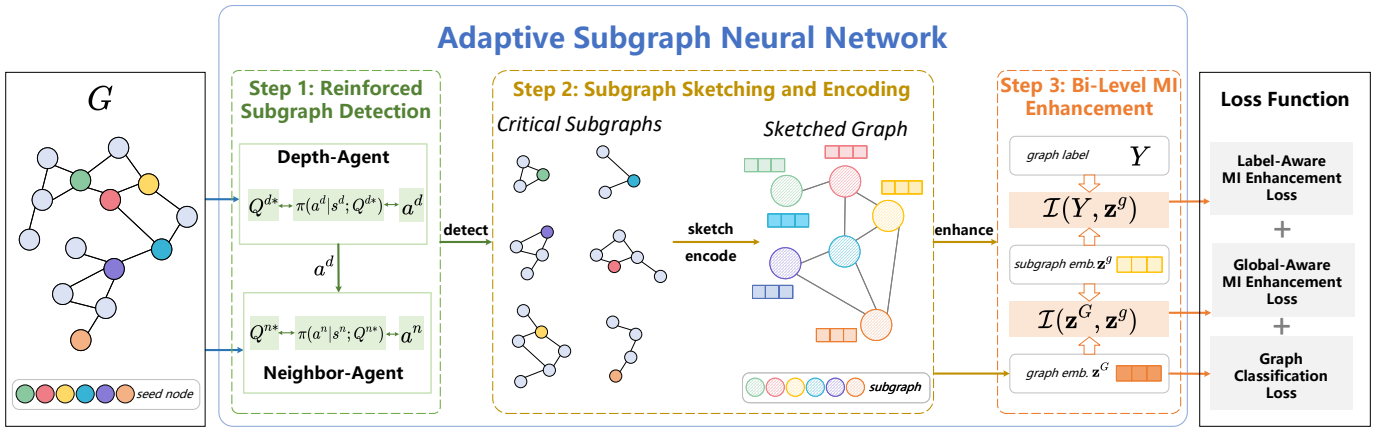


Fig. 2: An illustration of the AdaSNN architecture. Step-1: the Reinforced Subgraph Detection Module aims to detect critical subgraphs $\{g_i\}$ for an input graph $G$. Step-2: the critical subgraphs $\{g_i\}$ are used to contract a sketched graph $G^{ske}$ and encoded into subgraph representations $\{z_i^g\}$. Step-3: the Bi-Level MI Enhancement Mechanism further enhances the subgraph representations and uses them for classification.

Before providing a detailed introduction to AdaSNN, we first describe its pipeline in Fig. 1. Without loss of generality, we introduce this process by focusing on a specific graph $G$ in the given dataset $\mathbf{G} = \{G\}$. Briefly, using AdaSNN for graph classification includes three steps: (1) Critical Subgraph Detection, which detects critical subgraphs from the input graph; (2) Subgraph Sketching and Encoding, which builds a sketched graph upon the detected subgraphs and learns the subgraph encodings; and (3) Subgraph Representation Enhancement, which further enhances the subgraph representations and uses them for classification. Fig. 2 depicts AdaSNN's overall architecture.

**Step-1: Critical Subgraph Detection.** To tackle the challenges (i.e., data-specific, task-specific, arbitrary size, arbitrary shape, and absence of annotations) mentioned in Section 1, we develop a Reinforced Subgraph Detection Module, which learns to find critical subgraphs from the original graph with the feedback of the downstream task without supervision information. The detected subgraphs determine the local neighborhoods of the center nodes to further gather the information around them.

Directly sampling some nodes and taking the induced subgraphs as the critical subgraphs are difficult due to the arbitrary size and rich dependencies of graphs. We first sample a set of seed nodes $\{v_i\}_{i=1}^N$ as the center nodes of subgraphs, and then sample subgraphs around these seed nodes. Moreover, we decompose the subgraph sampling task as two sub-tasks (i.e., depth decision and neighbor decision), and design two agents (i.e., Depth-Agent and Neighbor-Agent) to solve them. Specifically, for every center node $v_i$, we first use a 2-layer GNN encoder to transform the node features into the node encodings as the initialization. Then we detect the subgraph around $v_i$:

$$d_i \Leftarrow \pi^d(G, v_i), g_i \Leftarrow \pi^n(G, v_i, d_i), \tag{1}$$

where $d_i$ is the subgraph depth for center node $v_i$, $g_i$ is the detected subgraph around $v_i$, $\pi^d$ and $\pi^n$ are the learned policy of Depth-Agent and Neighbor-Agent, respectively. The Reinforcement Subgraph Detection Module is introduced in detail in Section 3.3.

**Step-2: Subgraph Sketching and Encoding.** After obtaining critical subgraphs $\{g_i\}$, we contract them into a sketched graph $G^{ske} = (V^{ske}, E^{ske})$ and then use an attention mechanism to learn subgraph embeddings. Specifically, we treat every subgraph as a supernode in the sketched graph. The connectivity between $V^{ske}$ is determined by the number of common nodes of two subgraphs. When the number of common nodes in $g_i$ and $g_j$ is more than a constant $b_{com}$, $e_{i,j}$ will be added to the edge set of the sketched graph.

$$\begin{aligned} V^{ske} &= \{g_i\},\ \forall\ i = 1, 2, \cdots, N; \\ E^{ske} &= \{e_{i,j}\},\ \forall\ \left|V(g_i) \bigcap V(g_j)\right| > b_{com}. \end{aligned} \tag{2}$$

Then we learn the representations of subgraphs from two perspectives: intra-subgraph and inter-subgraph. For the intra-subgraph perspective, we feed each subgraph $g_i$ into a GNN-based encoder, $\mathcal{E}$, to obtain an embedding of nodes within subgraphs. Then the node representations $\mathbf{H}(g_i)$ of nodes within $g_i$ can be calculated by the following generalized equation:

$$\mathbf{H}(g_i) = \mathcal{E}(g_i) = \{\mathbf{h}_i(v_j) | v_j \in V(g_i)\}, \tag{3}$$

where $\mathbf{h}_i(v_j)$ is the node encodings. $\mathcal{E}(\cdot)$ can be formulated as a unified message-passing framework:

$$\begin{aligned} &\mathbf{h}_i^{(l+1)}(v_j) = \mathrm{U}^{(l+1)}\left(\mathbf{h}_i^{(l)}(v_j), \mathrm{AGG}\left(\mathrm{M}^{(l+1)}(\mathbf{h}_i^{(l)}(v_k))\right)\right), \\ &\forall\ v_k \in V(g_i)\ and\ v_k \in \mathcal{N}(v_j)), \end{aligned} \tag{4}$$

where $\mathbf{h}_i^{(l+1)}(v_j)$ is the node encoding of $v_i$ in the $(l+1)$-th GNN layer, $\mathrm{M}(\cdot)$ is the message generation, $\mathrm{AGG}(\cdot)$ is the message aggregation, and $\mathrm{U}(\cdot)$ is the updating function. Note that AdaSNN is flexible in terms of the GNN model used for the subgraph encoding step, various GNNs can be substituted for Eq. (3). Then we leverage a READOUT function to encode node encodings into a unified space for subgraph representations. After that, the hidden representations $\mathbf{h}_i^g$ of $g_i$ can be computed as follows:

$$\mathbf{h}_i^g = \mathrm{READOUT}\left(\{\mathbf{h}_i^L(v_j)\}, v_j \in V(g_i)\right), \tag{5}$$

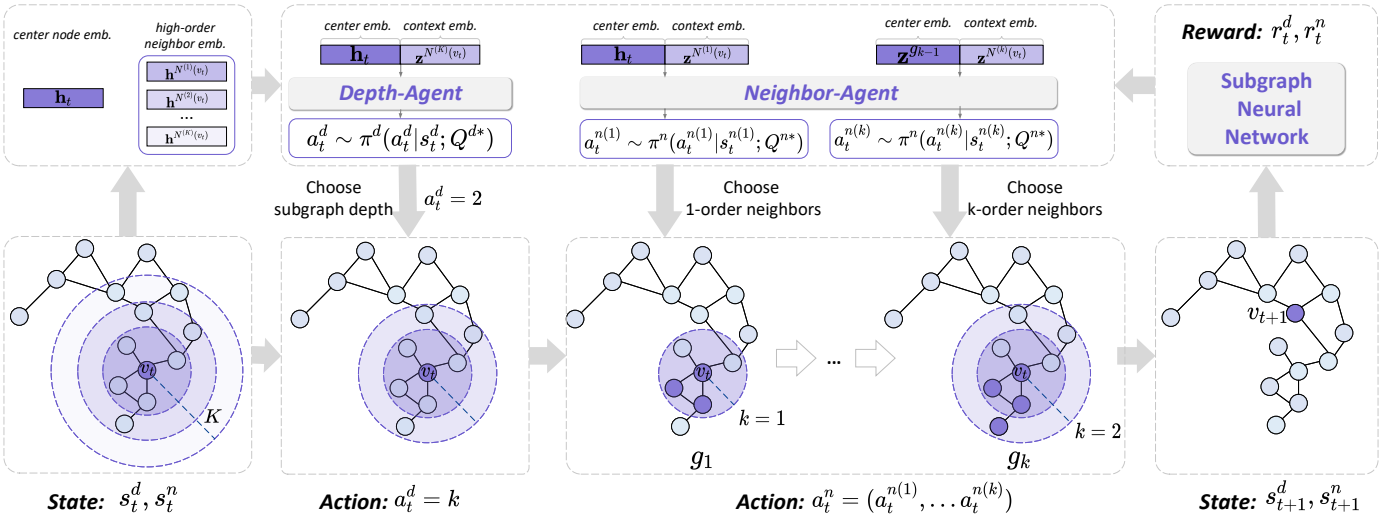


Fig. 3: An illustration of the reinforced subgraph detection module. For a given center node, a critical subgraph is detected in two phases: (1) For a center node $v_t$, Depth-Agent generates an action $a_t^d$ by policy $\pi^d$ to specify the number of hops for the current subgraph; (2) With the specified depth $k = a_t^d$, Neighbor-Agent generates actions $(a_t^{n(1)}, a_t^{n(2)}, \cdots, a_t^{n(k)})$ by policy $\pi^n$ to sample the member nodes of subgraph within $k$-hop neighbors of node $v_t$ hop by hop.

where $L$ denotes the number of layers in the GNN encoder $\mathcal{E}(\cdot)$, and the READOUT function can be any pooling operations and we use a simple sum pooling here.

Then we adopt an inter-subgraph attention mechanism to learn the interaction between subgraphs from their embedded vectors. Specifically, we calculate the attention coefficient $\alpha_{ij}$ of subgraph $g_i$ on $g_j$ by a multi-head attention mechanism as in [18]. The subgraph representations $\mathbf{z}^g = \{\mathbf{z}_i^g\}_{i=1}^N$ can be obtained by the attention guided information aggregation of its neighbor subgraph:

$$\mathbf{z}_i^g = \frac{1}{M}\sum_{m=1}^{M}\sum_{e_{ij}\in E^{ske}} \alpha_{ij}^m \mathbf{W}_{inter}^m \mathbf{h}_j^g, \tag{6}$$

where $\alpha_{ij}$ denotes the attention coefficient, $\mathbf{W}_{inter}$ denotes a trainable weight matrix, and $M$ denotes the number of attention heads.

**Step-3: Subgraph Representation Enhancement.** To obtain more discriminative representations, we design a Bi-Level Mutual Information (MI) Maximization Mechanism to further enhance the subgraph representations $\{\mathbf{z}_i^g\}_{i=1}^N$ in two perspectives: Global-Aware MI and Label-Aware MI.

The Global-Aware MI Maximization aims to encourage the subgraph representation to be aware of the global properties of the whole graph. Its objective is to maximize the MI between local subgraph representation $\mathbf{z}_i^g$ and the global graph representation $\mathbf{z}^G$ over the given dataset $\mathbf{G} = \{G\}$, where $G$ is the graph that $g_i$ belongs to.

$$\max \sum_{G\in\mathbf{G}} \frac{1}{N}\sum_{i=1}^{N} \mathcal{I}\left(\mathbf{z}_i^g, \mathbf{z}^G\right). \tag{7}$$

The Label-Aware MI Maximization aims to encourage the subgraph representations to be discriminative among graph samples. Its objective is to maximize MI between subgraph/label pair $(\mathbf{z}_i^g, Y)$ over the given dataset $\mathbf{G} = \{G\}$, where $Y$ is the label of the graph that $g_i$ belongs to.

$$\max \sum_{G\in\mathbf{G}} \frac{1}{N}\sum_{i=1}^{N} \mathcal{I}\left(\mathbf{z}_i^g, Y\right). \tag{8}$$

Ate last, the enhanced subgraph representations are summarized into a graph representation for the graph classification task. The Bi-Level MI Enhancement Mechanism is introduced in detail in Section 3.4.

### 3.3 Reinforced Subgraph Detection Module

In the Reinforced Subgraph Detection Module, we use a reinforcement learning algorithm to adaptively detect critical subgraphs of arbitrary size and shape. We model the proposed subgraph detection by a Finite Horizon Markov Decision Process (MDP) $\mathcal{M}(G, Y)$. As shown in Fig. 3, we sample a subgraph for each seed node in two phases:
(1) For a center node $v_t$, Depth-Agent generates an action $a_t^d$ according to the policy $\pi(s_t^d)$ to specify the depth $k$ for the subgraph around $v_t$;
(2) With the specified $k = a_t^d$, Neighbor-Agent generates actions $(a_t^{n(1)}, a_t^{n(2)}, \cdots, a_t^{n(k)})$ to sample the member nodes of subgraph within $k$-hop neighbors of $v_t$ hop by hop. Then we introduce the components of these two agents.

#### 3.3.1 *Depth-Agent.*

Depth-Agent chooses the depth of a subgraph for a given center node. The components $(\mathcal{S}^d, \mathcal{A}^d, \mathcal{R}^d, \mathcal{T}^d)$ of Depth-Agent are defined as:

- ***State*** $(\mathcal{S}^d)$. The state is defined as the center embedding of current center node $v_t$, which we denote as $\mathbf{h}_t$ for simplicity, and the context embedding of its K-hop neighbors $\mathbf{z}^{N^{(K)}(v_t)}$:

$$s_t^d = \left(\mathbf{h}_t, \mathbf{z}^{N^{(K)}(v_t)}\right), \tag{9}$$

where $\mathbf{z}^{N^{(K)}(v_t)}$ is the mean vector of all nodes in $v_t$'s K-hop neighbors. $K$ is a pre-defined hyper-parameter which determines the maximum subgraph depth.

- ***Action*** *($\mathcal{A}^d$).* The depth selection action of the Depth-Agent $a_t^d$ denotes the depth of the critical subgraph around the center node $v_t$, which is always a positive integer and $1 \leq a_t^d \leq K$.
- ***Reward*** *($\mathcal{R}^d$).* It is hard to sense the GNN's state due to its black-box nature, making the cumulative reward unavailable. For the reward function, we directly define a discrete reward function $r^d(s_t^d, a_t^d)$ based on the performance improvement on the downstream task as our purpose is to push the performance boundary. The reward function can be formulated as:

$$r^d(s_t^d, a_t^d) = \begin{cases} +0.5, & if\ \hat{Y}_t = Y, \\ -0.5, & if\ \hat{Y}_t \neq Y, \end{cases} \tag{10}$$

where $Y$ is the ground-truth label and $\hat{Y}$ is the predicted label at epoch $t$. In this work, Eq. (10) indicates the reward is positive if the graph classification accuracy with $(s_t^d, a_t^d)$ is higher than in $t-1$ and vice versa.

- ***Transition*** *($\mathcal{T}^d$).* After choosing the depth of the current subgraph, Depth-Agent samples the next center node in seed nodes as the next state $s_{t+1}^d$. Once the agent samples $N$ subgraphs for a graph instance, the process stops.

### 3.3.2 Neighbor-Agent.

Neighbor-Agent chooses the member nodes of a subgraph hop by hop with depth chosen by Depth-Agent. The components $(\mathcal{S}^n, \mathcal{A}^n, \mathcal{R}^n, \mathcal{T}^n)$ of Neighbor-Agent are defined as:

- ***State*** *($\mathcal{S}^n$).* The state is defined as the center embedding of current subgraph $\mathbf{z}^{g_{k-1}}$ and the context embedding $\mathbf{z}^{N^{(t)}(v_t)}$ of nodes in the $k$-th hop:

$$s_t^{n(k)} = \left(\mathbf{z}^{g_{k-1}}, \mathbf{z}^{N^{(k)}(v_t)}\right). \tag{11}$$

$\mathbf{z}^{g_{k-1}}$ is the mean vector of nodes in current $(k-1)$-hop subgraph and we set $\mathbf{z}^{g_1}$ as the center node $v_t$'s representation $\mathbf{h}_t$. $\mathbf{z}^{N^{(k)}(v_t)}$ is the mean vector of all nodes in $v_t$'s $k$-th hop neighbors.

- ***Action*** *($\mathcal{A}^n$).* With a specified depth integer $k = a_t^d$, the hop-by-hop neighbor sampling actions are denoted as $a_t^{n(1)}, a_t^{n(2)}, \cdots, a_t^{n(k)}$, where $a_t^{n(k)}$ denotes the action in the $k$-th hop of the Neighbor-Agent. Hence the subgraph sampling action can be decomposed into hierarchical actions $a_t^n = (a_t^{n(1)}, a_t^{n(2)}, \cdots, a_t^{n(k)})$.
- ***Reward*** *($\mathcal{R}^n$).* Similar with the Depth-Agent, we define the reward function $r^n(s_t^n, a_t^n)$ as the performance improvement on the downstream graph classification task to objectively measure the critical subgraph effect, with reward being:

$$r^n(s_t^n, a_t^n) = \begin{cases} +0.5, & if\ \hat{Y}_t = Y \\ -0.5, & if\ \hat{Y}_t \neq Y \end{cases} \tag{12}$$

- ***Transition*** *($\mathcal{T}^n$).* After choosing the member nodes by $a_t^n$, the Neighbor-Agent takes the representation of current subgraph and nodes in the next hop as $s_{t+1}^n$. Once the agent samples $k$-hop member nodes for the subgraph, the process stops.

**Algorithm 1:** Reinforced Subgraph Detection

**Input:** Graph $G = (V, X, E, A)$; maximum number of subgraph depth $K$; explosion probability $\epsilon$; reward window size $w$; memory buffers $\mathcal{B}^d$ and $\mathcal{B}^n$; training step $S$.
**Output:** Critical subgraphs $\{g_i\}$.

```
Initialize Q^d, Q^n, B^d and B^n;
// Subgraph Detection
s_1^d ← Eq. (9);
for t = 1, 2, ··· , N do
    Depth-Agent choose an action a_t^d by Q^d;
    for i = 1, 2, ··· , a_t^d do
        s_t^{n(i)} ← Eq. (11);
        Neighbor-Agent choose an action a_t^{n(i)} by Q^n;
        Expand the current subgraph;
    end
    Obtain r_t^d and r_t^n on the validation dataset by Eq. (10) and Eq. (12);
    Sample the next center node from the set of seed nodes;
    s_{t+1}^d ← Eq. (9);
    if B^d and B^n are full then
        Apply Algorithm 2 to train AdaSNN;
        Clear memory buffers B^d and B^n;
    end
    Store {s_t^d, a_t^d, s_{t+1}^d, r_t^d} into B^d;
    Store {s_t^n, a_t^n, s_{t+1}^n, r_t^n} into B^n;
end
// Optimization
for step = 1, 2, ··· , S do
    Optimize Q^d using the data in B^d by Eq. (13);
    Optimize Q^n using the data in B^n by Eq. (14).
end
```

In this work, we use the widely-used DQN [74], a model-free algorithm, to learn an optimal policy that can maximize the expected total reward overall steps starting from the current state. More advanced algorithms could also be employed in our framework. DQN [74] uses a deep neural network to approximate the state-action value $Q(s, a)$. Based the above reward functions Eq. (10) and Eq. (12), we train the Q functions for Depth-Agent and Neighbor-Agent, respectively:

$$Q^d(s^d, a^d) = \mathbb{E}_{s^{d'}}[\mathcal{R}(s^{d'}) + \gamma^d \max_{a^{d'}}(Q^d(s^{d'}, a^{d'})], \tag{13}$$

$$Q^n(s^n, a^n) = \mathbb{E}_{s^{n'}}[\mathcal{R}(s^{n'}) + \gamma^n \max_{a^{n'}}(Q^n(s^{n'}, a^{n'})], \tag{14}$$

where $\gamma^d$ and $\gamma^n \in [0, 1]$ denotes the discount factors of future reward. A *$\varepsilon$-greedy* policy with an explore probability $\varepsilon$ is adopted to obtain the policies $\pi^d$ and $\pi^n$:

$$\pi(a_t^d | s_t^d; Q^{d*}) = \begin{cases} \text{random action}, & \text{with probability } \varepsilon \\ \arg\max_{a_t^d} Q^{d*}(s_t^d, a_t^d), & \text{otherwise} \end{cases} \tag{15}$$

$$\pi(a_t^n | s_t^n; Q^{n*}) = \begin{cases} \text{random action}, & \text{with probability } \varepsilon \\ \arg\max_{a_t^n} Q^{n*}(s_t^n, a_t^n), & \text{otherwise} \end{cases} \tag{16}$$

It means that the RL agents explore new states by choosing a random action with probability $\varepsilon$ instead of choosing an action that can get the maximum future reward.

### 3.4 Bi-Level MI Enhancement Mechanism

Since the core idea of AdaSNN is extracting critical subgraphs as a representative abstraction of the graph, the critical subgraphs should be aware of global graph properties and be predictive for the downstream task. Formally, we designed a Bi-Level Mutual Information (MI) Enhancement mechanism from an information-theoretic perspective, aiming to enhance the subgraph representations on two levels:
(1) **Global-Aware MI**: MI between subgraph representation and graph representation;
(2) **Label-Aware MI**: MI between subgraph representation and graph label.

#### 3.4.1 Global-Aware MI Enhancement

To extract critical subgraphs at a global scale, we encourage the derived subgraph representations to preserve the global structure properties, rather than enforcing all properties contained in one overarching graph representation vector.

We use the Jensen-Shannon estimator [82] to maximize the MI over the local subgraph embedding and global embedding pairs in Eq. (7). The Jensen-Shannon estimator is approximately monotonic with the Kullback Leiber divergence [83], which can provide better and more stable results. The Global-Aware MI mechanism is performed in a contrastive way as the Jensen-Shannon estimator is based on discriminating the local/global pairs and negatively sampled pairs. In this work, we set the negative local/global pairs as a subgraph representation $\mathbf{z}^g$ with an alternative graph representation $\mathbf{z}^{G'}$. The global graph representation $\mathbf{z}^G$ is the summarization of the obtained subgraph-level embeddings via a $\mathrm{READOUT}(\cdot)$ function:

$$\mathbf{z}^G = \mathrm{READOUT}\left(\{\mathbf{z}_i^g\}_{i=1}^N\right). \tag{17}$$

We can adopt any permutation-invariant function (e.g, averaging and graph-level pooling) as the $\mathrm{READOUT}(\cdot)$ function. Specifically, we apply a simple mean strategy here. Then the global/local MI can be expressed as:

$$\begin{aligned}\mathcal{I}\left(\mathbf{z}^g, \mathbf{z}^G\right) :=& \mathbb{E}_{\mathbb{P}}\left[-\mathrm{sp}\left(-T\left(\mathbf{z}^g, \mathbf{z}^G\right)\right)\right] \\ &- \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}\left[\mathrm{sp}\left(T\left(\mathbf{z}^g, \mathbf{z}^{G'}\right)\right)\right],\end{aligned} \tag{18}$$

where $G$ is an input sample following the empirical probability distribution of the input space $\mathbb{P}$, $G'$ is a graph sampled from $\tilde{\mathbb{P}} = \mathbb{P}$ and $\mathrm{sp}(z) = \log(1+e^z)$ denotes the softplus function.

For the above objective, a discriminator $\mathcal{D}$ is introduced as in [83], which determines whether the input subgraph embedding and graph embedding are from the same graph. Then the Global-Aware MI Enhancement loss function $\mathcal{L}_{MI}^{global}$ can be formulated as a binary cross-entropy loss:

$$\begin{aligned}\mathcal{L}_{MI}^{global} = \frac{1}{N+N_{neg}}\Bigg(&\sum_{g_i\in G}^{N}\log\left(\mathcal{D}(\mathbf{z}_i^g, \mathbf{z}^G)\right) \\ &+ \sum_{g_j\in G}^{N_{neg}}\log\left(1-\mathcal{D}(\mathbf{z}_j^g, \mathbf{z}^{G'})\right)\Bigg),\end{aligned} \tag{19}$$

where $n_{neg}$ is the number of negative samples. As mentioned before, the Global-Aware MI Enhancement mechanism is performed in a contrastive way as it is based on discriminating local and global pairs and the negatively sampled pairs. The negative sampling strategy governs the specific kinds of captured structure information and is a critical implementation detail of contrastive methods. In our framework, the negative samples are generated in a batch-wise fashion by taking another graph instance in the current batch as $G'$. To investigate the impact of the negative sampling strategy, we design another negative sampling strategy $G'(V, X', A) = \mathcal{C}(G(V, X, A))$, which corrupt the original graph for negative samples. The corruption function $\mathcal{C}(\cdot)$ preserves original node sets $V$ and adjacency matrix $A$ whereas corrupts node feature matrix $X'$ by row-wise shuffling following the settings in [81]. We further analyze the impacts of two negative sampling strategies in Section 4.4.

#### 3.4.2 Label-Aware MI Enhancement

To extract task-relevant subgraphs, we encourage the detected subgraphs to be predictive of the graph label. We use the Label-Aware MI Enhancement Mechanism to maximize the relevance between the subgraph embeddings $\mathbf{z}^g$ and the graph label $Y$ in Eq. (8). According to the information theory, the mutual information $\mathcal{I}(Y, \mathbf{z}^g)$ between $\mathbf{z}^g$ and $Y$ is defined as:

$$\begin{aligned}\mathcal{I}(Y, \mathbf{z}^g) &= \int p(Y, \mathbf{z}^g)\log\frac{p(y|\mathbf{z}^g)}{p(y)}dY d\mathbf{z}^g \\ &= \int p(Y, \mathbf{z}^g)\log p(y|\mathbf{z}^g)dY d\mathbf{z}^g \\ &\quad - p(Y, \mathbf{z}^g)\log p(Y)dY d\mathbf{z}^g \\ &= \int p(Y, \mathbf{z}^g)\log p(Y|\mathbf{z}^g)dY d\mathbf{z}^g + H(Y).\end{aligned} \tag{20}$$

Examining Eq. (20), we can see that $H(Y)$ denotes the entropy of label set $Y$, which should be ignored during optimization. In practice, $p(Y, \mathbf{z}^g)$ can be approximated with an empirical distribution by sampling on training data. Then we can substitute the true posterior $p(Y|\mathbf{z}^g)$ with a variational approximation $q_\phi(Y|\mathbf{z}^g)$ for a tractable lower bound for $\mathcal{I}(Y, \mathbf{z}^g)$:

$$\begin{aligned}\mathcal{I}(Y, \mathbf{z}^g) &= \int p(Y, \mathbf{z}^g)\log p(Y|\mathbf{z}^g)dY d\mathbf{z}^g + H(Y) \\ &\geq \int p(Y, \mathbf{z}^g)\log q_\phi(Y|\mathbf{z}^g)dY d\mathbf{z}^g \\ &\quad + \mathrm{KL}(p(Y|\mathbf{z}^g)||q_\phi(Y|\mathbf{z}^g)).\end{aligned} \tag{21}$$

According to the non-negativity of Kullback Leiber divergence $\mathrm{KL}(p(Y|\mathbf{z}^g)||q_\phi(Y|\mathbf{z}^g) \geq 0$, then we have:

$$\begin{aligned}\mathcal{I}(Y, \mathbf{z}^g) &\geq \int p(Y, \mathbf{z}^g)\log q_\phi(Y|\mathbf{z}^g)dY d\mathbf{z}^g \\ &\quad + \mathrm{KL}(p(Y|\mathbf{z}^g)|q_\phi(Y|\mathbf{z}^g)) \\ &\geq \int p(Y, \mathbf{z}^g)\log q_\phi(Y|\mathbf{z}^g)dY d\mathbf{z}^g \\ &\approx \frac{1}{N}\sum_{N}^{i=1}\log q_\phi(Y_i|\mathbf{z}_i^g) \\ &=: \mathcal{L}_{MI}^{label}(\mathbf{z}^g, Y).\end{aligned} \tag{22}$$

**Algorithm 2:** Training process of AdaSNN

**Input:** Graphs with labels $\{G = (V, X, E, A), Y\}$; memory buffers $\mathcal{B}^d$ and $\mathcal{B}^n$; Number of epochs $epoch_{max}$.
**Output:** Predicted graph label $\hat{Y}$

```
for e = 1, 2, ..., epoch_max do
    // Critical Subgraph Detection
    Obtain detected critical subgraphs {g_i} from B^d and B^n;
    // Subgraph Sketching and Encoding
    Contract the sketched graph G_ske ← G;
    Calculate the intra-subgraph representation h_i^g by Eq. (5);
    Calculate the inter-subgraph representations z^g by Eq. (6);
    Readout the graph representations z^G by Eq. (17);
    // Subgraph Representation Enhancement
    Sample negative samples z^{G'};
    Calculate the Global-Aware MI loss L_MI^global by Eq.(19);
    Calculate the Label-Aware MI loss L_MI^label by Eq.(22);
    // Optimization
    Calculate the overall loss L by Eq.(23);
    Update parameters by descending L;
end
```

Eq. (22) indicates that $\mathcal{I}(Y, \mathbf{z}^g)$ can be maximized by minimizing the classification loss $\mathcal{L}_{MI}^{label}$ between $\mathbf{z}^g$ and $Y$. Intuitively, minimizing $\mathcal{L}_{MI}^{label}$ encourages the subgraph representation to be relevant to the graph label, making $\mathbf{z}^g$ predictive to $Y$. In practice, we set $\mathcal{L}_{MI}^{label}$ as the cross-entropy loss.

### 3.5 Optimization of AdaSNN

We define the overall loss $\mathcal{L}$ of AdaSNN as the combination of the supervised graph classification loss $\mathcal{L}_{cls}$, the Global-Aware MI loss $\mathcal{L}_{MI}^{global}$ and the Label-Aware MI loss $\mathcal{L}_{MI}^{label}$:

$$\mathcal{L} = \mathcal{L}_{cls} + \beta_1 \mathcal{L}_{MI}^{global} + \beta_2 \mathcal{L}_{MI}^{label}, \tag{23}$$

where $\mathcal{L}_{cls} = CE(\mathbf{z}^G, Y)$ is the cross-entropy loss for graph classification, $\beta_1$ and $\beta_2$ control the contribution of the Global-Aware MI Enhancement and Label-Aware MI Enhancement, respectively. In doing so, AdaSNN can preserve both local and global graph structure properties as well as rich discriminative intermediate subgraph representations.

Based on the aforementioned study, Algorithm 2 outlines the training process of the proposed AdaSNN. To make the training process more stable, we use a memory buffer mechanism as in [65]. Specifically, we construct two buffers $\mathcal{B}^d$ and $\mathcal{B}^n$ for the Depth-Agent and the Neighbor-Agent. The obtained states and actions are stored in the buffers. When the buffer is full, we randomly sample the experience data in the buffer to optimize the corresponding Q function by Eq. (13) and Eq. (14) and then clear the buffer.

TABLE 3: Statistics of datasets.

| Dataset | # Graphs | # Classes | Max. $\|V\|$ | Avg. $\|V\|$ | Node Features |
|---|---|---|---|---|---|
| MUTAG | 188 | 2 | 28 | 17.9 | 7 |
| PTC | 344 | 2 | 64 | 14.29 | 18 |
| PROTEINS | 1,113 | 2 | 620 | 39.1 | 3 |
| IMDB-B | 1,000 | 2 | 136 | 19.8 | 136 |
| IMDB-M | 1,500 | 3 | 89 | 13 | 89 |
| REDDIT-B | 2,000 | 2 | 3,782 | 429.6 | 1 |
| OGBG-PPA | 158,100 | 37 | 300 | 243.4 | 300 |

## 4 Experimental Analysis

This section empirically evaluates the proposed AdaSNN focusing on the following research questions:

- **Q1.** How does AdaSNN perform in improving graph learning quality and robustness? (Section 4.2)
- **Q2.** How does the Reinforcement Subgraph Detection Module influence the performance of AdaSNN? (Section 4.3)
- **Q3.** How does the Bi-Level MI Enhancement Mechanism influence the performance of AdaSNN? (Section 4.4)
- **Q4.** Can AdaSNN be trained stably with the complex RL based subgraph detection strategy? (Section 4.5)
- **Q5.** Can AdaSNN detect subgraphs with prominent patterns and provide insightful interpretations? (Section 4.7)

### 4.1 Experimental Setups

#### 4.1.1 Datasets

We consider the commonly-used benchmark datasets for graph classification. The dataset statistics are summarized in Table 3.

- **Bioinformatics datasets**: MUTAG [85] contains 188 compounds marked according to whether it has a mutagenic effect on a bacterium, where the node features are the one-hot encodings of atom types; PTC [86] contains 344 organic molecules marked according to their carcinogenicity on male mice, where the node features are the one-hot encodings of atom types; PROTEINS [87] contains 1,113 proteins marked as enzyme or non-enzyme, where the node features are the one-hot encodings of the amino acid types. OGBG-PPA [88] contains 158,100 undirected protein association neighborhoods extracted from the protein-protein association networks marked according to what taxonomic group the graph originates from, where the node features are the one-hot encodings of node degrees.
- **Social network datasets**: IMDB-B [89] and IMDB-M [89] are movie collaboration datasets marked according to the genre an ego-network belongs to, where the node features are the one-hot encodings of node degree; REDDIT-B [89] contains 2,000 online discussion thread graphs between users marked according to their subreddit, where the node features are the normalized node degrees.

#### 4.1.2 Baselines

We consider a number of state-of-the-art baselines, including graph neural networks, graph pooling methods, critical structure detection methods, a graph transformer, and a graph reinforcement learning method to demonstrate the effectiveness and robustness of AdaSNN.

TABLE 4: Graph classification results: "average accuracy±standard deviation (rank)". (**bold**: best; underlined: runner-up.)

| Model | Dataset | | | | | | Avg. Rank |
|---|---|---|---|---|---|---|---|
| | MUTAG | PTC | PROTEINS | IMDB-B | IMDB-M | REDDIT-B | |
| GCN [50] | 74.3±11.0 (12) | 57.2±5.6 (8) | 74.3±11.0 (4) | 70.7±3.7 (15) | 50.4±5.6 (8) | 73.6±4.5 (15) | 10.3 |
| GAT [18] | 73.8±7.4 (13) | 56.4±7.4 (10) | 71.4±4.0 (12) | 71.3±4.2 (10) | 50.7±7.4 (5) | 73.1±2.6 (16) | 11.0 |
| GIN [19] | 82.5±6.8 (6) | 56.1±3.3 (13) | 70.7±5.6 (13) | 73.2±4.8 (5) | 50.1±7.4 (9) | 85.4±3.0 (10) | 9.3 |
| DiffPool [42] | 83.9±9.7 (4) | 53.1±10.9 (16) | 72.7±4.6 (6) | 70.9±5.3 (13) | 48.7±6.5 (12) | 88.8±2.3 (4) | 9.2 |
| ASAPool [45] | 74.3±7.7 (11) | 57.6±6.6 (6) | 72.1±4.3 (11) | 71.5±4.4 (9) | 51.3±6.6 (3) | 88.9±2.5 (3) | 7.1 |
| Top-K Pool [44] | 71.8±1.1 (15) | 56.1±1.1 (12) | 70.5±1.5 (14) | 71.2+2.5 (11) | 47.2±1.1 (15) | 75.5±4.6 (12) | 13.2 |
| SortPool [23] | 84.4±14.1 (2) | 56.7±6.8 (9) | 74.7±4.4 (3) | 71.2±4.7 (12) | 49.9±6.8 (10) | 85.1±5.0 (11) | 7.8 |
| SAGPool [43] | 71.8±3.1 (16) | 55.2±1.5 (15) | 70.3±0.8 (15) | 70.7±2.0 (14) | 48.5±5.5 (13) | 74.1±2.8 (14) | 14.5 |
| EdgePool [84] | 75.9±7.7 (10) | 58.4±5.4 (4) | 72.3±4.4 (9) | 72.8±4.4 (8) | 49.5±5.4 (11) | 89.7±2.0 (2) | 7.3 |
| SAT [56] | 83.8±1.7 (5) | 58.6±2.1 (3) | 73.3±2.7 (5) | 72.9±1.9 (6) | 50.4±2.8 (6) | 88.6±1.8 (6) | 5.2 |
| PolicyGNN [65] | 79.8±2.8 (8) | 59.4±1.8 (2) | 72.6±1.1 (7) | 73.7±3.1 (3) | 48.2±4.3 (14) | 85.7±2.1 (7) | 6.8 |
| Ego-CNN [28] | 73.4±1.1 (14) | 55.5±2.8 (14) | 68.1±4.9 (16) | 72.7±2.8 (7) | 50.7±1.2 (4) | 74.5±3.3 (13) | 11.3 |
| SIB [27] | 83.9±6.4 (3) | 57.3±2.8 (7) | 74.9±5.1 (2) | 73.7±7.0 (4) | 51.6±4.8 (2) | 85.7±3.5 (9) | 4.5 |
| CAL [62] | 82.2±7.2 (7) | 58.2±8.7 (5) | 72.6±3.3 (8) | 68.7±5.0 (16) | 47.2±4.7 (16) | 88.7±1.5 (5) | 9.5 |
| SUGAR [31] | 78.7±7.9 (9) | 56.3±1.9 (11) | 72.2±4.5 (10) | 73.8+3.4 (2) | 50.4±4.4 (7) | 87.6±2.5 (8) | 7.8 |
| **AdaSNN(Ours)** | **87.2±5.0 (1)** | **60.2±6.4 (1)** | **76.5±2.6 (1)** | **74.2±2.5 (1)** | **51.9±4.7 (1)** | **89.7±1.4 (1)** | 1.0 |

- **Graph neural networks**: GCN [50] performs spectral graph convolutions on the graph adjacency matrix; GAT [18] leverages the attention mechanism to assign different weights to different neighborhoods; GIN [19] is a message-passing model designed by the Weisfeiler-Lehman graph isomorphism test.
- **Graph pooling methods**: DiffPool [42] learns a differentiable soft cluster assignment for nodes and maps clusters to supernodes as a coarsened graph layer by layer; ASAPool [45] utilizes an attention mechanism to capture the nodes' importance and pools the subgraphs to a coarsen graph by a learnable sparse soft cluster assignment; Top-k Pool [44] adaptively selects some critical nodes to form a smaller subgraph based on their importance vectors; SortPool [23] sorts graph nodes in a consistent order and designs a localized graph convolution model trained on the node sequences; SAGPool [43] uses a self-attention mechanism to retain some nodes and drop the others; EdgePool [84] learns a localized and sparse pooling transform by edge contraction.
- **Critical structure detection methods**: Ego-CNN [28] employs the ego-convolutions at each layer and detects precise critical structures in an ego-centric way by stacking layers; SIB [27] extends the general information bottleneck principle to graph data for the subgraph recognition problem; SUGAR [31] first samples some subgraphs and then uses a DQN algorithm to choose top-k critical subgraphs as the representative abstraction of the whole graph. CAL [62] proposes the Causal Attention Learning (CAL) strategy to discover the critical subgraphs by estimating the causal and shortcut features via an attention module.
- **Graph transformer**: SAT [56] is the only existing graph transformer method that incorporates the structure explicitly. It incorporates the structural information into the original self-attention by extracting a k-hop subgraph or a k-subtree rooted at each node before computing the attention.
- **Graph reinforcement learning method**: PolicyGNN [65] uses a reinforcement learning algorithm to determine the depth of aggregations for each node adaptively.

#### 4.1.3 Implementation Details

For all GNN-based baselines and graph pooling baselines, we adopt the implementations from the PyTorch Geometric Library[2] in all experiments. For the remaining baselines (Ego-CNN[3], SIB[4], SAT[5], PolicyGNN[6], CAL[7]), we use the source codes provided by the authors and we use the GIN as the backbone. We perform a hyper-parameter search for common parameters of all models on the validation set and follow the settings in their original papers in terms of other hyper-parameters.

For our AdaSNN, we use GIN [19] with 16 hidden units as the subgraph encoder in Eq.(3) and the mean pooling operation as the READOUT function in Eq. (17). The number of detected subgraphs $N$ is chosen according to the graph size. We construct a 5-layers of *MLP* with (64, 128, 256, 128, 64) hidden units for $Q$ function in *DQN*. For the *$\varepsilon$-greedy* policy, we set up a linear scheduler, where $\varepsilon$ decades from 1.0 to 0.2 with a discount factor 0.95 every 100 steps. The loss coefficients $\alpha_1$ and $\alpha_2$ are chosen by a grid search strategy.

### 4.2 Overall Evaluation (Q1)

This subsection evaluates AdaSNN on both graph classification and graph denoising tasks to verify AdaSNN's ability to improve graph learning quality and robustness.

#### 4.2.1 Graph Classification

We performed 10-fold cross-validation on MUTAG, PTC, PROTEINS, IMDB-B, IMDB-M and REDDIT-B[8]. We report the accuracy, standard deviation, and rank of every method

2 https://github.com/rusty1s/pytorch_geometric
3 https://github.com/rctzeng/EgoCNN
4 https://github.com/Samyu0304/graph-information-bottleneck-for-Subgraph-Recognition
5 https://github.com/BorgwardtLab/SAT
6 https://github.com/datamllab/Policy-GNN
7 https://github.com/yongduosui/CAL
8 We follow the protocol in https://github.com/rusty1s/pytorch_geometric/tree/master/benchmark/kernel

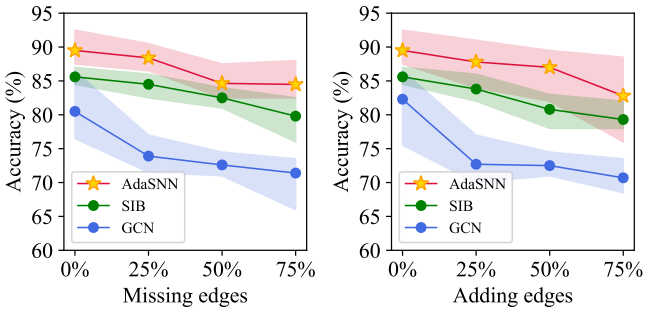


Fig. 4: Test accuracy (± standard deviation) in the edge attack scenarios on REDDIT-B.

TABLE 5: Graph classification results on OGBG-PPA: "average accuracy±standard deviation". (**bold**: best).

| Method | GIN | ASAPool | SAT | SIB | SUGAR | AdaSNN |
|---|---|---|---|---|---|---|
| Acc.±Std | 68.12±1.2 | 71.24±3.8 | 74.17±1.8 | 72.88±5.0 | 73.25±6.8 | **74.31±1.8** |

in Table 4, where the best ones are in bold and the runner-ups are underlined.

As shown in Table 4, AdaSNN consistently outperforms fifteen baselines on six datasets. AdaSNN achieves more gains consistently compared to selection-based pooling methods (e.g., SortPool [44], SAGPool [43]) and the global denoising method SIB [27], supporting our intuition of information denoising in subgraph-level. The graph transformer SAT [56] is the third among all baselines, showing the effectiveness of capturing the importance of subgraphs. AdaSNN outperforms SAT because it can exploit subgraphs of arbitrary size and shape while SAT is built upon fixed k-hop subgraphs. The graph reinforcement learning method Policy-GNN [65] also does not perform as well as AdaSNN because it only learns the order of the subgraphs rather than the more concrete structures. Compared the recent critical subgraph detection method Ego-CNN [28], SIB [27] and CAL [62], AdaSNN also shows advantage in classification accuracy, which achieves 3.3% and 4.0% improvements in terms of average accuracy on MUTAG and REDDIT-B than SIB [27], respectively. Specifically, our AdaSNN and SIB [27] perform better than Ego-CNN [28]. It may be because Ego-CNN [28] is limited to the enumerated simple ego-networks, while our AdaSNN can capture more complex subgraphs of varied sizes and shapes.

We also evaluate the proposed AdaSNN with competitive baselines on a larger dataset ODBG-PPA. The results are shown in Table 5, where the best results are shown in bold. AdaSNN achieves the best performance.

*Altogether, the proposed AdaSNN shows very promising results on graph classification task against the advanced methods.*

#### 4.2.2 Graph Denoising

To evaluate the robustness of AdaSNN, we generate synthetics datasets by deleting or adding edges in the REDDIT-B dataset. Specifically, for each graph, we randomly remove (if edges exist) or add (if no such edges) 25%, 50%, and 75% edges. We report the results of the vanilla GCN, the most competitive baseline SIB and our method in Fig. 4, where the solid line is the mean accuracy and the shaded region is the standard deviation over 5 runs.

As shown in Fig. 4, the accuracy of all methods dropped with the increasing noise degrees and GCN suffers the most. Since SIB and AdaSNN can find the most predictive critical subgraphs, they show better robustness than GCN, and our AdaSNN achieves better results with less performance degradation. We notice that other baseline methods show similar trends as SIB, not shown to avoid cluttering the figure.

*Benefit from the critical structure detection, the proposed AdaSNN is resistant to the task-irrelevant information and shows better robustness.*

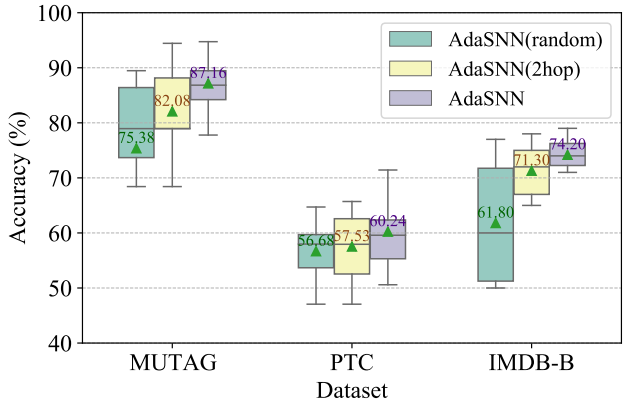


Fig. 5: AdaSNN with different subgraph detection strategies. (▲ denotes the mean accuracy value)

### 4.3 Analysis of the Reinforced Subgraph Detection Module (Q2)

This subsection analyzes the impact of the Reinforced Subgraph Detection Module in terms of its effectiveness, stability, and parameter sensitivity.

#### 4.3.1 Impact of the Reinforced Subgraph Detection Module

To demonstrate the effectiveness of the Reinforced Subgraph Detection Module, we introduce the following two variants of AdaSNN:

- **AdaSNN(2hop)**, which takes fixed 2-hop subgraphs as critical subgraphs;
- **AdaSNN(random)**, which takes subgraphs with randomly chosen depth as critical subgraphs.

The results of AdaSNN and two variants on three datasets are summarized in Figure 5. Since we take the critical subgraphs as a representative abstraction of the whole graph, the quality of detected subgraphs directly determines the quality of graph representation. As shown in Figure 5, the mean accuracy of AdaSNN with the Reinforced Subgraph Detection Module is higher than the other variants on three datasets and achieves at most 5.08% improvement over AdaSNN(2hop). The consistent improvement supports the intuition behind our critical subgraph detection approach.

We also compare AdaSNN with the preliminary version SUGAR [31] to demonstrate the effectiveness of critical subgraphs of arbitrary shapes and sizes. Remind that SUGAR

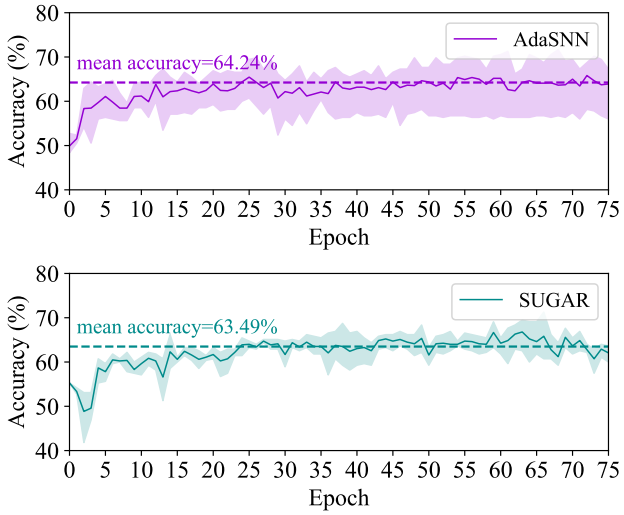


Fig. 6: Training accuracy dynamics of AdaSNN and SUGAR on PTC dataset.

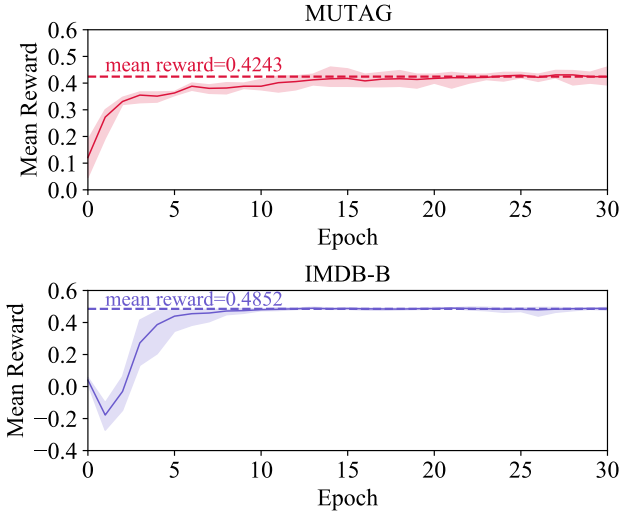


Fig. 7: Mean reward of the Reinforcement Subgraph Detection Module.

is a two-phase framework that samples subgraphs by fixed rules and then selects critical ones, where the reinforcement learning algorithm is used to decide the number of critical subgraphs. The critical subgraphs detected by SUGAR are constrained by the sampling rules. We plot the training process of AdaSNN and SUGAR in Fig. 6, where the enclosed shadowed area is the range of accuracies in three training runs, the middle solid line is the mean accuracy, and the dashed horizontal line is the mean accuracy value in the last 10 epochs.

As we can observe, the training process of AdaSNN is not as stable as SUGAR, which maybe because of the more complex reinforcement learning algorithm for critical subgraph detection. The mean accuracy of AdaSNN achieves about 0.78% improvement than SUGAR, supporting our intuition behind the critical subgraph detection.

*This indicates the proposed Reinforced Subgraph Detection Module is effective to improve the performance of graph classification by extracting more meaningful and predictive subgraphs.*

#### 4.3.2 Reinforcement Learning Process.

Since the RL algorithm in the Reinforced Subgraph Detection Module and the subgraph neural network are trained jointly, their updating and convergence are indeed important. Fig. 7 shows the training process of AdaSNN on MUTAG and IMDB-B over five runs. The shadowed area is enclosed by the minimum and maximum rewards and the middle solid line is the mean reward over five runs. The dashed horizontal line denotes the mean reward in the last 10 epochs.

As we can observe, the mean reward converges to 0.4243 on MUTAG and 0.4852 on IMDB-B with a stable learning curve within 30 epochs, which means that the framework can predict true labels based on the detected subgraph.

*This indicates the proposed Reinforced Subgraph Detection Module can be trained with the subgraph neural network stably and converges fast.*

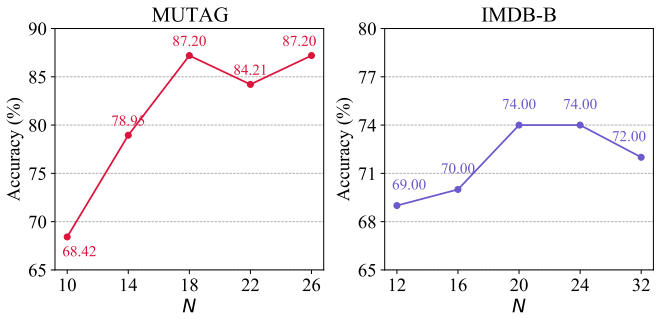


Fig. 8: Parameter sensitivity of the number of detected critical subgraphs $N$.

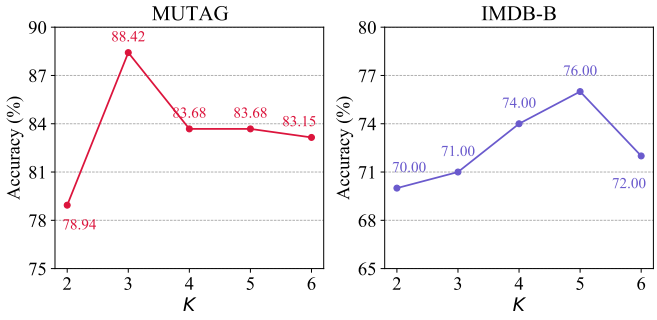


Fig. 9: Parameter sensitivity of the maximum subgraph depth $K$.

#### 4.3.3 Parameter Sensitivity of the Number of Subgraphs and the Maximum Subgraph Depth

This subsection analyzes the impact of the number of detected subgraphs $N$ and the maximum subgraph depth $K$.

In Fig. 8, we evaluate the proposed AdaSNN with different subgraph numbers $n$ from 10 to 26 on MUTAG (left) and from 12 to 32 on IMDB-B (right). Note that the average node number of graph instances in MUTAG and IMDB-B is 18 and 20, respectively. It shows a similar trend on

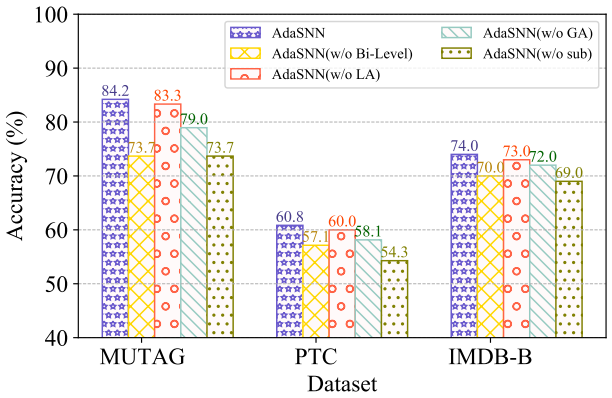


Fig. 10: AdaSNN with different MI enhancement mechanisms.

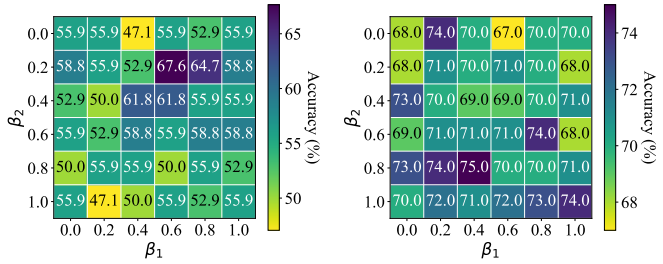


Fig. 11: Parameter sensitivity of loss coefficients $\beta_1$ and $\beta_2$ on PTC (left) and IMDB-B (right).

both datasets that the performance first increases and then bumps with the increase number of subgraphs. Although AdaSNN does not give satisfactory results with a small number of subgraphs, it is found that considering more subgraphs obviously helps to improve performance. It can also be observed that when the number of subgraphs is larger than the average degree, the performance of AdaSNN was is not significantly improved.

In Fig. 9, we evaluate the proposed AdaSNN with different maximum subgraph depths $K$ from 2 to 6 on MUTAG (left) and IMDB-B (right). $K$ determines the agent state as well as the range of the action set. As shown in Fig. 9, AdaSNN achieves the best performance with $K$=3 in MUTAG and $K = 5$ in IMDB-B. This indicates that the critical subgraphs in social networks have a higher order than in bioinformatics graphs. It is reasonable because the diameters of most of the basic functional blocks in molecules are around 3 and the long-range dependency also plays an important role in social networks [90].

*This indicates that AdaSNN can obtain competitive performance when the detected critical subgraphs can cover the important functional building blocks of the graph.*

### 4.4 Analysis of the Bi-Level MI Enhancement Mechanism (Q3)

This subsection analyzes the impact of the Bi-Level MI Enhancement Mechanism in terms of its effectiveness, negative sampling strategy, and parameter sensitivity.

#### 4.4.1 Impact of the Bi-Level MI Enhancement Mechanism

To verify the effectiveness of the Bi-Level MI Enhancement Mechanism, we introduce the following three variants of AdaSNN:

- **AdaSNN(w/o Bi-Level)**: AdaSNN without the Bi-Level MI Enhancement mechanism, whose loss is $\mathcal{L} = \mathcal{L}_{cls}$;
- **AdaSNN(w/o LA)**: AdaSNN without Label-Aware MI Enhancement, i.e., with Global-Aware MI enhancement, whose loss is $\mathcal{L} = \mathcal{L}_{cls} + \beta_1 \mathcal{L}_{MI}^{global}$;
- **AdaSNN(w/o GA)**: AdaSNN without the Global-Aware MI Enhancement, i.e., with Label-Aware MI enhancement, whose loss is $\mathcal{L} = \mathcal{L}_{cls} + \beta_2 \mathcal{L}_{MI}^{label}$.
- **AdaSNN(w/o sub)**: AdaSNN without the subgraph detection and sketching, which uses the GIN as the graph encoder and directly maximizes the Global-Aware Mutual Information between nodes and the graph and the Label-Aware Mutual Information between nodes and the graph label.

We show the performance of AdaSNN and its three variants on MUTAG, PTC, and IMDB-B in Fig. 10. We can observe that AdaSNN with Bi-Level MI Enhancement Mechanism shows better performance than the other three variants. In addition, AdaSNN(w/o LA) consistently outperforms AdaSNN(w/o GA) and shows a difference of about one percent from AdaSNN on all three datasets, indicating that the Label-Aware MI plays a more important role in enhancing the subgraphs to be critical and discriminative. AdaSNN(w/o sub) enforces the single node preserving both graph properties and label information and shows unsatisfied performance, indicating the effectiveness of extracting subgraphs explicitly.

We also analyze the parameter sensitivity of the Bi-Level MI loss coefficients $\beta_1$ and $\beta_2$ in Eq. (23) on PTC (left) and IMDB-B (right) datasets. $\beta_1$ controls the contribution of the Global-Aware MI loss $\mathcal{L}_{MI}^{global}$ and $\beta_2$ controls the contribution of the Label-Aware MI loss $\mathcal{L}_{MI}^{label}$. As we can observe in Fig. 11, AdaSNN performs best on PTC dataset when $\beta_1$ is from 0.6 to 0.8 and $\beta_2$ is from 0.2 to 0.4. For the PTC dataset, the Global-Aware MI loss $\mathcal{L}_{MI}^{global}$ plays a major role in the Bi-Level MI enhancement mechanism, which suggests that making the subgraphs contain the overall graph properties benefits more for social networks. For the IMDB-B dataset, the Label-Aware MI loss $\mathcal{L}_{MI}^{label}$ plays a major role in the Bi-Level MI enhancement mechanism, indicating that making the subgraphs contain the label information benefits more for social networks. *This indicates that the Bi-Level MI Enhancement mechanism can give informative self-supervision to AdaSNN so that subgraph representations can be discriminative towards other graph instances. The proposed Bi-Level MI enhancement mechanism can capture different critical information for different types of datasets through two enhancement ways.*

#### 4.4.2 Impact of the Negative Sampling Strategy

The negative sampling strategy for $G'$ is a critical implementation detail of Global-Aware MI Enhancement, which governs the specific kinds of captured structure information. We further analyze the impacts of the negative sampling strategy and negative sampling ratio. As mentioned in Section 3.4.1, AdaSNN samples an alternative graph $G'$ from

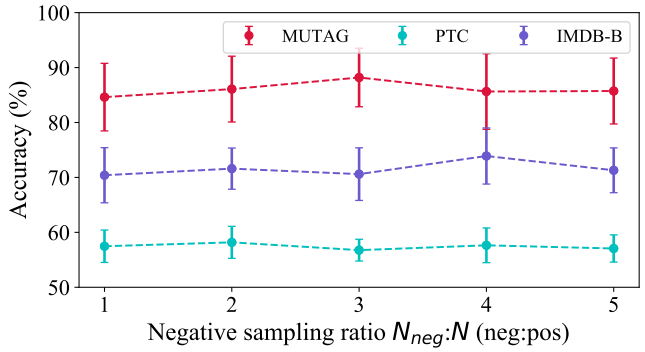


Fig. 12: Parameter sensitivity of the negative sampling ratio.

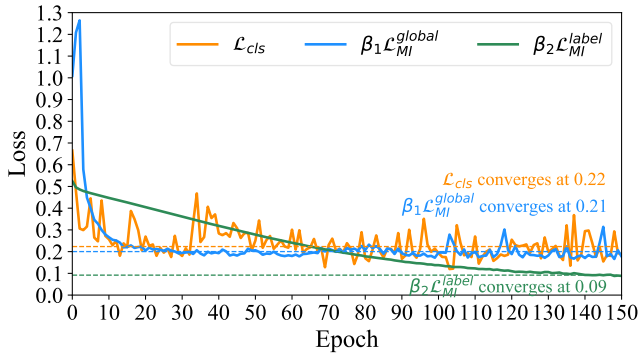


Fig. 13: Training stability of AdaSNN on MUTAG dataset.

TABLE 6: Performance comparison of AdaSNN(Corrupt) and AdaSNN.

| Method / Dataset | MUTAG | PTC | PROTEINS | IMDB-B |
|---|---|---|---|---|
| AdaSNN(Corrupt) | 78.1±6.1 | 55.2±9.8 | 74.2±5.8 | 70.1±6.0 |
| **AdaSNN** | **87.2±5.0** | **60.2±6.4** | **76.5±2.6** | **74.2±2.5** |

$\tilde{\mathbb{P}} = \mathbb{P}$. The negative samples are generated in a batch-wise fashion by taking another graph instance in the batch as the alternative graph $G'$. We also design another negative sampling strategy for $G'$:

- **AdaSNN(Corrupt)**, which constructs negative samples by graph corruption $G'(V, X', A) = \mathcal{C}(G(V, X, A))$. The corruption function $\mathcal{C}(\cdot)$ corrupts node feature matrix by row-wise shuffling and preserves original $V$ and $A$.

The comparison results of AdaSNN and AdaSNN(MI Corrupt) are shown in Table 6. We can observe that AdaSNN (i.e., sampling another graph from the batch) consistently outperforms AdaSNN(MI Corrupt) (i.e., corrupting the graph). One possible reason is that the graph corruption will make $G'$ lose important structure properties and can only give weak supervision.

*This indicates that the negative sampling strategy should be carefully designed so that subgraph representations can be discriminative towards other graph instances.*

#### 4.4.3 Parameter Sensitivity of the Negative Sampling Ratio

The Global-Aware MI enhancement mechanism aims to maximize the mutual information between critical subgraphs and graphs in a contrastive way. The common belief is that competitive contrastive methods need a large number of negative samples [91]. Fig. 12 shows the performances of AdaSNN with different negative sampling ratios $n_{neg} : n$ in Eq. (19). With the increase of negative samples, the performance of AdaSNN varies within three percent on three datasets. The larger negative sampling ratio seems to not boost the performance of AdaSNN significantly. A possible reason is that the negative samples are drawn for every detected critical subgraph, which already provides sufficient self-supervision even with a small negative sampling ratio. Considering the computational overhead caused by a large number of negative samples, we set the negative sampling ratio to 1 in other experiments.

*This indicates the effectiveness of the proposed AdaSNN in terms of finding meaningful and effective critical subgraphs which can act as hard negative samples in the self-supervised MI enhancement mechanism.*

### 4.5 Training Stability of AdaSNN (Q4)

The core idea of AdaSNN is critical subgraph abstraction by the Reinforced Subgraph Detection Module. Combining the GNN and reinforcement learning algorithm is indeed complex and challenging. In addition, the estimation of mutual information in the self-supervised Bi-Level MI enhancement mechanism always suffers from unstable training. In this subsection, we analyze the convergence of AdaSNN.

We show the training process of AdaSNN on MUTAG dataset in Fig. 13, including the graph classification loss $\mathcal{L}_{cls}$, the Global-Aware MI loss $\mathcal{L}_{MI}^{global}$ and the Label-Aware MI loss $\mathcal{L}_{MI}^{label}$. Here we set $\beta_1 = 0.7$ and $\beta_2 = 0.1$ in Eq. (23) and set $learning - rate = 0.01$. The dash lines in Fig. 13 indicate the mean value in the last 10 epochs when AdaSNN converges. We can see that the Global-Aware MI loss $\mathcal{L}_{MI}^{global}$ quickly converges within 20 epochs and the Label-Aware MI loss $\mathcal{L}_{MI}^{label}$ converges steadily within 150 epochs. The graph classification loss $\mathcal{L}_{cls}$ bumps but still decreases during the training process. It may be because the shapes of detected critical subgraphs change very quickly with the update of the reinforcement learning agents in the Reinforced Subgraph Detection Module.

*Overall speaking, even with the complex combination of RL algorithms, self-supervised MI enhancement mechanism, and GNN, the proposed AdaSNN can be trained stably.*

### 4.6 Complexity and Time Analysis

In this section, we will analyze the computational complexity and consumption of AdaSNN. The overall time complexity of on graph of AdaSNN is $O(N\bar{b}^K + N\bar{b}^2 + N^2)$, where $N$ is the number of subgraphs, $\bar{b}$ is the average node degree and $K$ is the maximum subgraph depth. Specifically, the time complexity of the Reinforced Subgraph Detection Module is $O(N\bar{b}^K)$. The 2-layer GIN encoder takes $O(N\bar{b}^2)$. The subgraph sketching takes $O(N^2)$. For the Bi-Level MI Enhancement Mechanism, loss calculation takes $O(N)$. The overall memory complexity of AdaSNN is $O(N\bar{b}^K)$. In our experiments, $N$ is selected from 5 to 20 according to the graph size and $K$ is set to 4 or 5. In addition, we count the time consumption of 10 epochs and report the

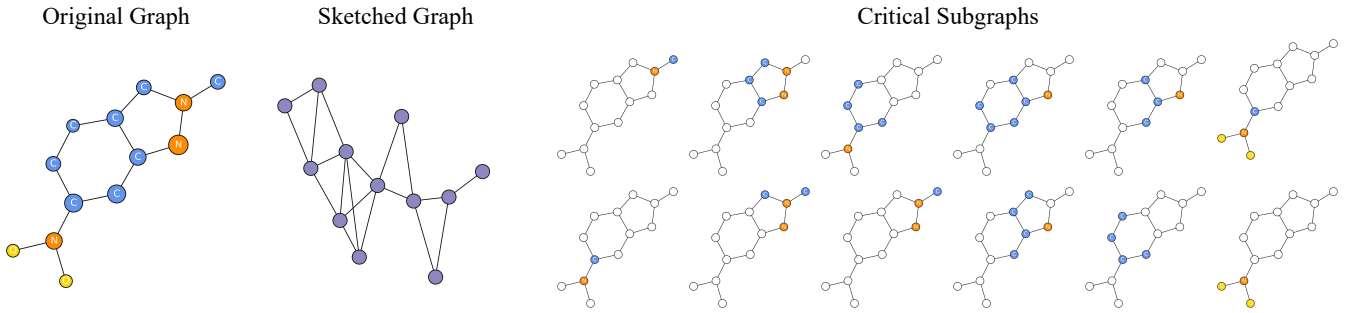


Fig. 14: Result visualization of AdaSNN on MUTAG dataset (Graph ID: 111).

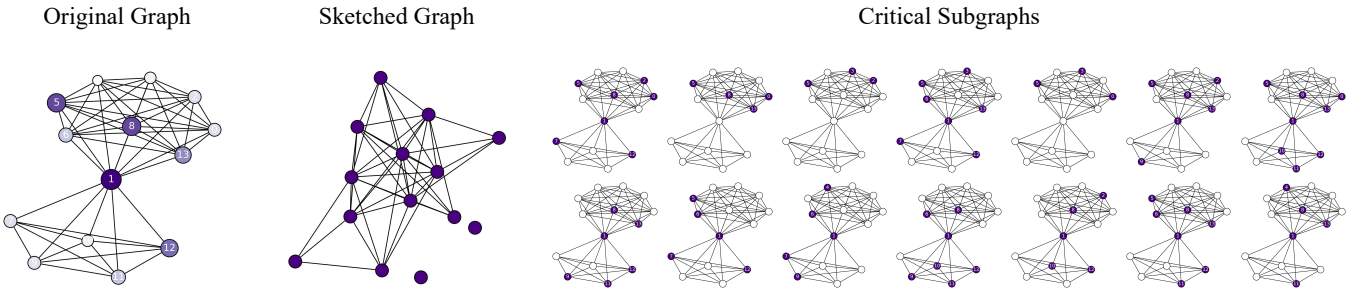


Fig. 15: Result visualization of AdaSNN on IMDB-B dataset (Graph ID: 292).

average time of one epoch in Table 7. All experiments run on the same multi-node GPU cluster, where each node in the cluster consists of a 64-core Intel Xeon E5-2680v4 CPU @ 2.40GHz with 512GB RAM and an NVIDIA V100 GPU with 32GB RAM. AdaSNN shows comparable efficiency with the substructure-aware methods (SAT and SIB) and the graph reinforcement learning method (PolicyGNN) when achieving the best performance.

TABLE 7: Statistics of the average training time of one epoch (in second).

| Method / Dataset | MUTAG | PTC | PROTEINS | IMDB-B | IMDB-M | REDDIT-B |
|---|---|---|---|---|---|---|
| DiffPool | 1.63 | 4.83 | 6.52 | 6.45 | 7.84 | 35.27 |
| EdgePool | 0.44 | 0.61 | 2.78 | 2.65 | 3.28 | 21.23 |
| SAT | 3.69 | 5.64 | 14.30 | 15.26 | 19.28 | 46.78 |
| PolicyGNN | 4.88 | 3.50 | 10.42 | 12.18 | 14.99 | 58.26 |
| SIB | 1.69 | 2.95 | 10.13 | 8.39 | 11.97 | 41.56 |
| **AdaSNN** | 5.13 | 6.68 | 21.81 | 17.71 | 23.78 | 40.18 |

### 4.7 Visualization (Q5)

This subsection visualizes the results to provide further insight of the changes brought by AdaSNN intuitively.

To examine the capability of AdaSNN on interpreting the property of graphs, we further visualize some results in Fig. 14 and Fig. 15, respectively. In both Fig. 14 and Fig. 15, the left part is the original graph, where the node size is proportional to its frequency in the critical subgraphs. For MUTAG dataset in Fig. 14, the node color denotes the node's element type as labeled on it. For IMDB dataset in Fig. 15, the shade of node color is proportional to node's frequency in the critical subgraphs. The middle part is the sketched graph with the sketch threshold $b_{com} = 1$ and $b_{com} = 2$ on MUTAG and IMDB-B, respectively. The right part is the critical subgraphs detected by the Reinforced Subgraph Detection Module.

**Chemical Compounds.** In the MUTAG dataset, there are 188 molecule graphs labeled by their mutagenic effect on bacteriums. The mutagenic molecules' main determinant is nitro elements and the nitro group $NO_2$ connected to a carbon ring [85]. As we can observe in Fig. 14, the proposed AdaSNN successfully detects the nitro group and the carbon rings. Besides, the sketched graph can reflect the higher-order interactions between them.

**Social Interactions.** IMDB-B is a movie collaboration dataset consisting of 1,000 ego-networks of actors and actresses who played roles in movies in IMDB. The ego-networks are labeled as the Action genre or Romance genre, where the nodes represent actors/actresses and the edges represent the concurrence of actors in a movie. As in Fig. 15, AdaSNN can find the close relations between a few nodes (i.e., some important actors) rather than treating all nodes equally. Besides, most of the detected subgraphs contain the bridging node between the two communities in the original graph. The sketched graph also reflects the interactions between actor groups.

*The visualization results suggest that the proposed AdaSNN can detect discriminative subgraphs and has great promise to provide interpretability for the learned results.*

## 5 Conclusion

This work proposed AdaSNN, a novel graph representation framework by critical subgraph detection and abstraction. AdaSNN detects critical subgraphs adaptively by a Reinforcement Subgraph Detection Module and reconstructs a sketched graph by contracting critical subgraphs. It fur-

ther enhances the subgraph representations to be discriminative by a Bi-Level Mutual Information Enhancement Mechanism. In this way, AdaSNN preserves both local and global properties hierarchically. AdaSNN provides explicit subgraph-level explanations rather than node or edge level, which can provide good interpretability and insight into graph analysis. Extensive experiments show the effectiveness and advantages of the proposed AdaSNN.

In the future, we will study how to extend more advanced RL algorithms to our models for more efficient critical subgraph detection. In addition, we aim to extend the RL-guided subgraph neural network to more graph mining applications, such as subgraph classification, graph interpretation, etc.

## REFERENCES


[1] J. W. Essam and M. E. Fisher, "Some basic definitions in graph theory," *Reviews of Modern Physics*, vol. 42, no. 2, p. 271, 1970.

[2] M. Culp and G. Michailidis, "Graph-based semisupervised learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 1, pp. 174–179, 2007.

[3] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *IJCNN*, vol. 2, no. 2005, 2005, pp. 729–734.

[4] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[5] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *ICLR*, 2016.

[6] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[7] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[8] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[9] T. Pourhabibi, K.-L. Ong, B. H. Kam, and Y. L. Boo, "Fraud detection: A systematic literature review of graph-based anomaly detection approaches," *Decision Support Systems*, vol. 133, p. 113303, 2020.

[10] T. Ma, C. Xiao, J. Zhou, and F. Wang, "Drug similarity integration through attentive multi-view graph auto-encoders," in *IJCAI*, 2018, pp. 3477–3483.

[11] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, "Semi-supervised graph classification: A hierarchical graph perspective," in *WWW*, 2019.

[12] L. Cai, J. Li, J. Wang, and S. Ji, "Line graph neural networks for link prediction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[13] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE transactions on Big Data*, vol. 6, no. 1, pp. 3–28, 2018.

[14] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, pp. 31–42, 1976.

[15] G. Bouritsas, F. Frasca, S. P. Zafeiriou, and M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[16] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, vol. 70, 2017, pp. 1263–1272.

[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2017.

[19] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[20] M. Zhu, X. Wang, C. Shi, H. Ji, and P. Cui, "Interpreting and unifying graph neural networks with an optimization framework," in *Proceedings of the Web Conference 2021*, 2021, pp. 1215–1226.

[21] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *NeurIPS*, 2015, pp. 2224–2232.

[22] Z. Wang and S. Ji, "Second-order pooling for graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[23] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018.

[24] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[25] Q. Xuan, J. Wang, M. Zhao, J. Yuan, C. Fu, Z. Ruan, and G. Chen, "Subgraph networks with application to structural feature space expansion," *IEEE transactions on Knowledge and Data Engineering*, 2019.

[26] T. Zuo, Y. Qiu, and W.-S. Zheng, "Neighbor combinatorial attention for critical structure mining," in *IJCAI*, 2020.

[27] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, and R. He, "Graph information bottleneck for subgraph recognition," in *ICLR*, 2021.

[28] R.-C. Tzeng and S.-H. Wu, "Distributed, egocentric representations of graphs for detecting critical structures," in *ICML*, 2019, pp. 6354–6362.

[29] J. B. Lee, R. A. Rossi, X. Kong, S. Kim, E. Koh, and A. Rao, "Graph convolutional networks with motif-based attention," in *CIKM*, 2019, pp. 499–508.

[30] H. Peng, J. Li, Q. Gong, Y. Ning, S. Wang, and L. He, "Motif-matching based subgraph-level attentional convolutional network for graph classification." in *AAAI*, 2020, pp. 5387–5394.

[31] Q. Sun, J. Li, H. Peng, J. Wu, Y. Ning, P. S. Yu, and L. He, "Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism," in *The Web Conference*, 2021.

[32] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *ICML*, 2016, pp. 2014–2023.

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

[34] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *ICLR*, 2013.

[35] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NeurIPS*, 2016, pp. 3844–3852.

[36] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *AAAI*, 2018, pp. 3546–3553.

[37] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Graph neural networks with convolutional arma filters," *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[38] A. Taheri, K. Gimpel, and T. Berger-Wolf, "Learning graph representations with recurrent neural network autoencoders," *KDD Deep Learning Day*, 2018.

[39] S. Ivanov and E. Burnaev, "Anonymous walk embeddings," in *ICML*, 2018, pp. 2191–2200.

[40] S. Verma and Z. L. Zhang, "Graph capsule convolutional neural networks," *arXiv preprint arXiv:1805.08090*, 2018.

[41] Z. Xinyi and L. Chen, "Capsule graph neural network," in *ICLR*, 2019.

[42] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS*, 2018.

[43] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *ICML*, vol. 97, 2019, pp. 3734–3743.

[44] H. Gao and S. Ji, "Graph u-nets," in *ICML*, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97, 2019, pp. 2083–2092.

[45] E. Ranjan, S. Sanyal, and P. P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations." in *AAAI*, 2020, pp. 5470–5477.

[46] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *SIGKDD*, 2019, pp. 723–731.

[47] H. Gao, Y. Liu, and S. Ji, "Topology-aware graph pooling networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4512–4518, 2021.

[48] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, 2011.

[49] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *SIGKDD*, 2015.

[50] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2016.

[51] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *ICML*. PMLR, 2020, pp. 1725–1735.

[52] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[53] C. Yang, M. Liu, V. W. Zheng, and J. Han, "Node, motif and subgraph: Leveraging network functional blocks through structural convolution," in *ASONAM*. IEEE, 2018, pp. 47–52.

[54] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 877–28 888, 2021.

[55] Z. Wu, P. Jain, M. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13 266–13 279, 2021.

[56] D. Chen, L. O'Bray, and K. Borgwardt, "Structure-aware transformer for graph representation learning," in *ICML*. PMLR, 2022, pp. 3469–3489.

[57] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, and R. He, "Recognizing predictive substructures with subgraph information bottleneck," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[58] E. Alsentzer, S. G. Finlayson, M. M. Li, and M. Zitnik, "Subgraph neural networks," in *NeurIPS*, 2020.

[59] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning theory and kernel machines*. Springer, 2003, pp. 129–143.

[60] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *ICDM*, 2005, pp. 8–pp.

[61] F. Monti, K. Otness, and M. M. Bronstein, "Motifnet: a motif-based graph convolutional network for directed graphs," in *2018 IEEE Data Science Workshop (DSW)*. IEEE, 2018, pp. 225–228.

[62] Y. Sui, X. Wang, J. Wu, M. Lin, X. He, and T.-S. Chua, "Causal attention for interpretable and generalizable graph classification," in *SIGKDD*, 2022, pp. 1696–1705.

[63] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[64] N. Mingshuo, C. Dongming, and W. Dongqi, "Reinforcement learning on graph: A survey," *arXiv preprint arXiv:2204.06127*, 2022.

[65] K.-H. Lai, D. Zha, K. Zhou, and X. Hu, "Policy-gnn: Aggregation optimization for graph neural networks," in *SIGKDD*, 2020, pp. 461–471.

[66] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *ICML*. PMLR, 2018, pp. 1115–1124.

[67] S. Zhu, I. Ng, and Z. Chen, "Causal discovery with reinforcement learning," in *ICLR*, 2020.

[68] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," in *ICML*. PMLR, 2021, pp. 12 241–12 252.

[69] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[70] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 315–324.

[71] H. Peng, R. Zhang, Y. Dou, R. Yang, J. Zhang, and P. S. Yu, "Reinforced neighborhood selection guided multi-relational graph neural networks," *ACM Transactions on Information Systems (TOIS)*, vol. 40, no. 4, pp. 1–46, 2021.

[72] L. Wang, W. Yu, W. Wang, W. Cheng, W. Zhang, H. Zha, X. He, and H. Chen, "Learning robust representations with graph denoising policy network," in *ICDM*. IEEE, 2019, pp. 1378–1383.

[73] H. Peng, R. Zhang, S. Li, Y. Cao, S. Pan, and P. Yu, "Reinforced, incremental and cross-lingual event detection from social messages," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[74] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[75] Z. Zhong, C.-T. Li, and J. Pang, "Reinforcement learning enhanced heterogeneous graph neural network," *arXiv preprint arXiv:2010.13735*, 2020.

[76] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," *arXiv preprint arXiv:1904.09981*, 2019.

[77] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, vol. 69, no. 6, p. 066138, 2004.

[78] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *ICLR*, 2019.

[79] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in *WWW*, 2020, pp. 259–270.

[80] P. Wang, Y. Fu, Y. Zhou, K. Liu, X. Li, and K. A. Hua, "Exploiting mutual information for substructure-aware graph representation learning." in *IJCAI*, 2020, pp. 3415–3421.

[81] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax." in *ICLR*, 2019.

[82] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," in *NeurIPS*, 2016, pp. 271–279.

[83] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," in *ICLR*, 2019.

[84] F. Diehl, T. Brunner, M. T. Le, and A. Knoll, "Towards graph pooling by edge contraction," in *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*, 2019.

[85] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[86] H. Toivonen, A. Srinivasan, and C. Helma, "Statistical evaluation of the predictive toxicology challenge," *Bioinformatics*, 2003.

[87] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[88] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22 118–22 133, 2020.

[89] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015, pp. 4292–4293.

[90] D. Lyu, Y. Yuan, L. Wang, X. Wang, and A. Pentland, "Investigating and modeling the dynamics of long ties," *Communications Physics*, vol. 5, no. 1, pp. 1–9, 2022.

[91] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Transactions on Knowledge and Data Engineering*, 2021.


**Jianxin Li** is currently a Professor with the School of Computer Science and Engineering, and Beijing Advanced Innovation Center for Big Data and Brain Computing in Beihang University. His current research interests include social networks, machine learning, big data, and trustworthy computing. Dr. Li has published research papers in top-tier journals and conferences, including the IEEE TKDE, TDSC, JAIR, ACM TOIS, TKDD, KDD, AAAI, and WWW.

**Qingyun Sun** is currently a Ph.D. candidate at the School of Computer Science and Engineering, and Beijing Advanced Innovation Center for Big Data and Brain Computing at Beihang University. Her research interests include machine learning and graph mining. She has published several papers on Web Conference, AAAI, ICDM, CIKM, etc.

**Hao Peng** is currently an Assistant Professor at the School of Cyber Science and Technology in Beihang University. His research interests include representation learning, social network mining, and reinforcement learning. To date, Dr. Peng has published over 70+ research papers in top-tier journals and conferences, including the IEEE TPAMI, TKDE, TPDS, TNNLS, TASLP, JAIR, ACM TOIS, TKDD, and Web Conference.

**Beinig Yang** is currently a M.S. candidate at the Beijing Advanced Innovation Center for Big Data and Brain Computing in Beihang University. He received a B.S. degree at the College of Computer Science and Technology in Harbin Engineering University. His research interests include machine learning and social network mining.

**Jia Wu** is an ARC DECRA Fellow at the School of Computing, Macquarie University, Sydney, Australia. He received the PhD from the University of Technology Sydney, Australia. His current research interests include data mining and machine learning. He has published 100+ refereed research papers in the above areas such as TPAMI, TKDE, TNNLS, TMM, NIPS, KDD, ICDM, IJCAI, AAAI and WWW. Dr. Wu was the recipient of SDM'18 Best Paper Award in Data Science Track and IJCNN'17 Best Student Paper Award. He currently serves as an Associate Editor of ACM *Transactions on Knowledge Discovery from Data* (TKDD). He is a Senior Member of IEEE.

**Phillp S. Yu** is a Distinguished Professor and the Wexler Chair in Information Technology at the Department of Computer Science, University of Illinois at Chicago. Before joining UIC, he was at the IBM Watson Research Center, where he built a world-renowned data mining and database department. He is a Fellow of the ACM and IEEE. Dr. Yu was the Editor-in-Chiefs of ACM TKDD (2011-2017) and IEEE TKDE (2001-2004).