南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

Test-time Adaptation on Graphs via Adaptive Subgraph-based Selection and Regularized Prototypes

Yusheng Zhao [1] Qixin Zhang [2] Xiao Luo [# 3] Junyu Luo [1] Wei Ju [1] Zhiping Xiao [# 4] Ming Zhang [# 1]

ICML 2025

Background

Test time adaptation (TTA) in Euclidean data:

- **Self-training** often involve selecting reliable data for efficient self-training and aim to construct self-supervising signals via contrastive learning or pseudo-labeling.
- **Data-centric methods** aim to construct virtual samples related to the test distribution as data augmentation.

Domain adaptation in non-Euclidean data:

Some prior works investigate the problem of attempt to transfer the knowledge from the labeled training graphs to unlabeled test graphs via domain discrepancy minimization.

However, these methods typically require labeled source graphs to perform domain alignment, which is often impractical when **facing data privacy issues** (Tan et al., 2023).

Motivation

(1) How to overcome the label scarcity of test graphs more reliably in the face of structural shifts?

(2) How to utilize the knowledge from unknown training graphs and the information from unlabeled test graphs more effectively? (avoid easily overfit and lose the knowledge in training graph)

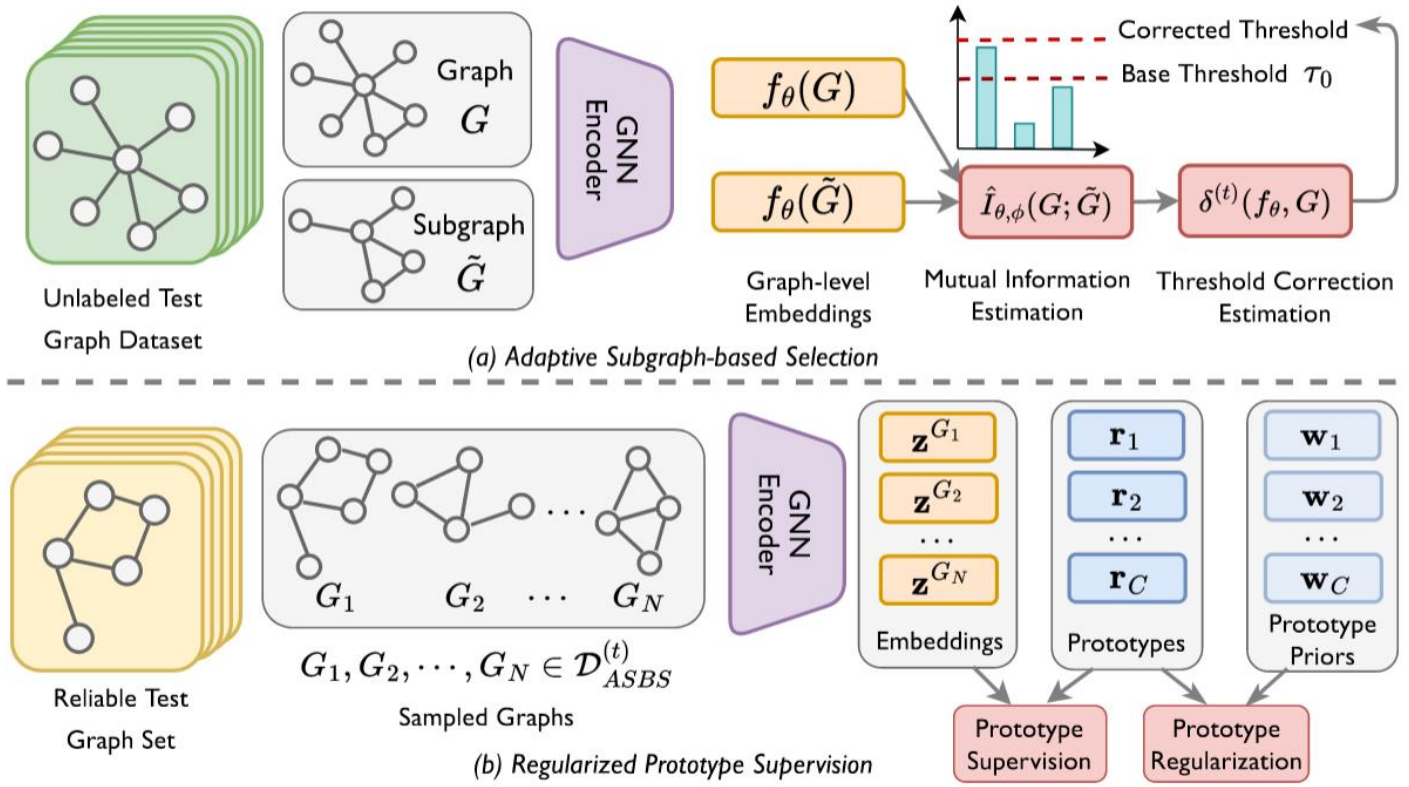


Figure 1. The overall framework of the proposed method. We first select reliable test graphs in the unlabeled test graph dataset using adaptive subgraph-based selection (ASBS), where we utilize mutual information to generate structure-aware, individual-level thresholds. Subsequently, we utilize these graphs for self-training with regularized prototype supervision (RPS), where the prototypes are regularized by prior knowledge and used for supervising the learned embedding of test graphs.

Formulation

$$G = (V, \boldsymbol{A}, \boldsymbol{X})$$

V is the set of nodes

$$A \in \mathbb{R}^{|V| \times |V|}$$

$$X \in \mathbb{R}^{|V| \times d^f}$$

$$\mathcal{D}^{tr} = \{(G_i^{tr}, y_i^{tr})\}_{i=1}^{N_{tr}}$$

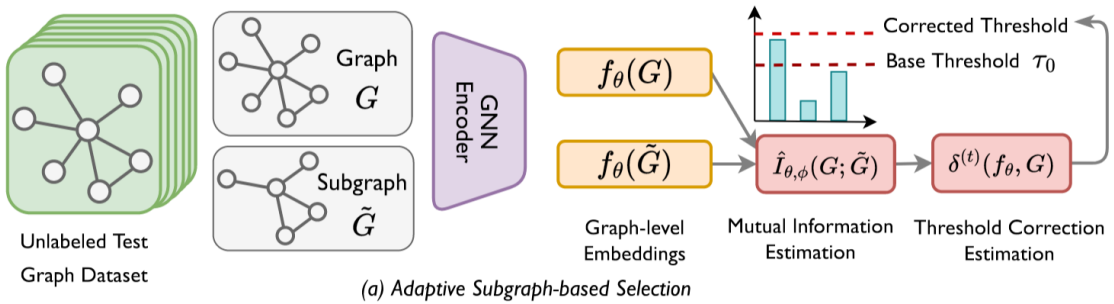
$$\mathcal{D}^{te} = \{G_j^{te}\}_{j=1}^{N_{te}}$$

$$\boldsymbol{z}^G = f_\theta(G)$$

$$\boldsymbol{p}^G = \text{softmax}(\boldsymbol{W}^T \boldsymbol{z}^G)$$

$$\boldsymbol{p}^G = \mathcal{M}(G)$$

Method/Adaptive Subgraph-based Selection



(a) Adaptive Subgraph-based Selection

$$s_{conf}^{G} = \max_{i \in \{1,\cdots,C\}} p_i^G, \quad \mathcal{D}^{conf} = \{G \in \mathcal{D}^{te} | s_{conf}^{G} > \tau\},$$

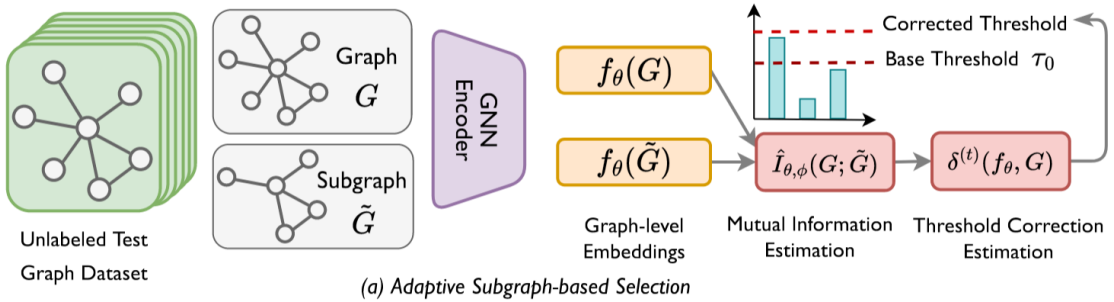
The threshold can be a shared threshold or class-wise threshold.

As the model can be inaccurate in exploring the unlabeled test data, especially when it comes to graphs with complex inherent structures.

Therefore, it is reasonable to use different selection thresholds for graph structures with different levels of complexity.

Graphs with complex structures are hard to learn for the model, and the model tends to be inaccurate. Thus, **a higher threshold should be used, and vice versa**.

Method/Adaptive Subgraph-based Selection



(a) Adaptive Subgraph-based Selection

Instead of a shared threshold, we adopt a graph-specific, individual-level threshold

$$\tau^G = \tau_0 + \delta(G, f_\theta),$$

δ measures the model's ability to yield an accurate prediction from G.

we propose to use mutual information (MI) between graph G and its subgraph

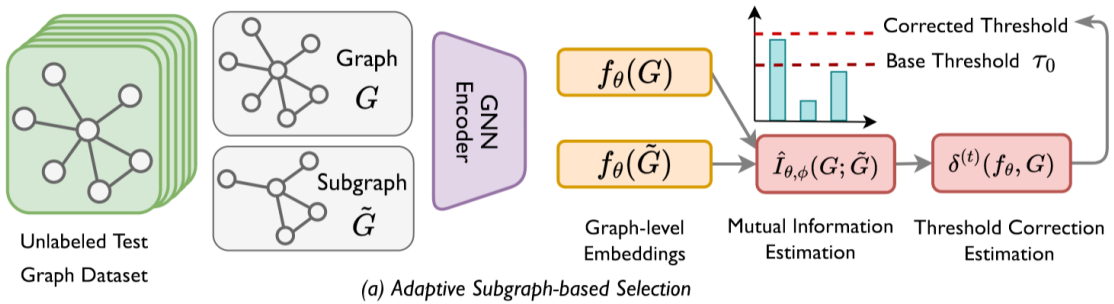
$$\tilde{G} = (V_{idx}, \boldsymbol{A}_{idx,idx}, \boldsymbol{X}_{idx,:})$$

$$I_{\theta,\phi}(G;\tilde{G}) = -\mathbb{E}_{G\sim\mathcal{D}^{te}}[\mathrm{sp}(-g_\phi(f_\theta(G), f_\theta(\tilde{G}))]$$

```python
class DiscriminatorMI(nn.Module):
    def __init__(self, embed_dim):
        super(DiscriminatorMI, self).__init__()
        self.fc_1 = nn.Sequential(
            nn.Linear(embed_dim, embed_dim),
            nn.ReLU(inplace=True),
            nn.Linear(embed_dim, embed_dim)
        )
        self.fc_2 = nn.Sequential(
            nn.Linear(embed_dim, embed_dim),
            nn.ReLU(inplace=True),
            nn.Linear(embed_dim, embed_dim)
        )

    def forward(self, embeddings_1, embeddings_2):
        emb_1 = self.fc_1(embeddings_1) + embeddings_1
        emb_2 = self.fc_2(embeddings_2) + embeddings_2
        score = torch.sum(emb_1 * emb_2, dim=1)
        return score
```

Method/Adaptive Subgraph-based Selection



(a) Adaptive Subgraph-based Selection

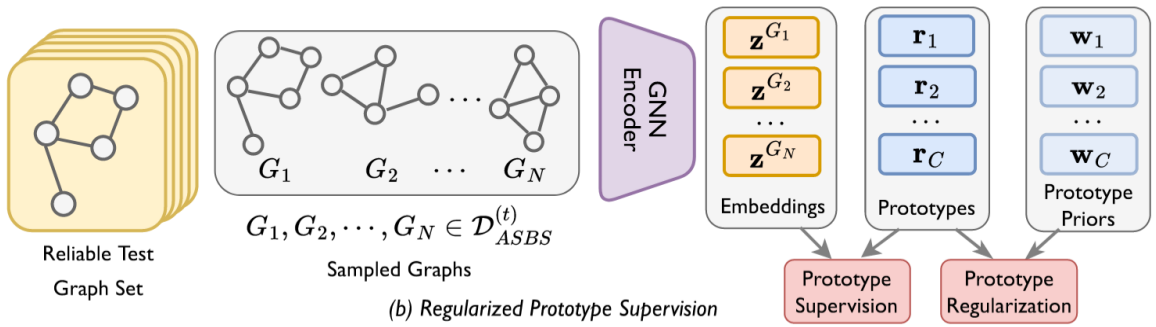
$$\tau^G = \tau_0 + \delta(G, f_\theta),$$

$$\delta(G, f_\theta) = -\omega \hat{I}_{\theta,\phi}(G; \tilde{G}),$$

$$\delta^{(t)}(G, f_\theta) = \omega \sum_{i=0}^{t-1} \beta(1-\beta)^i \hat{I}^{(t-i)}_{\theta,\phi}(G; \tilde{G}),$$

leads to a stable estimation of mutual information.
β controlling the speed of updates

$$\mathcal{D}^{(t)}_{ASBS} = \{G \in \mathcal{D}^{te} | s^G_{conf} > \tau^G = \tau_0 + \delta^{(t)}(G, f_\theta)\}.$$



(b) Regularized Prototype Supervision

initialize them with the weight matrix of the last layer of the model

$$R = [r_1, r_2, \cdots, r_C] \in \mathbb{R}^{d \times C}$$

$$\boldsymbol{R}^* = \underset{\boldsymbol{R}}{\operatorname{argmax}} \log p(\boldsymbol{R}|G),\ G \in \mathcal{D}^{(t)}_{ASBS}.$$

$$\log p(\boldsymbol{R}|G) = \log p(\boldsymbol{R}) + \log p(G|\boldsymbol{R}) + C_0, \qquad \boldsymbol{R} \sim \mathcal{N}(\boldsymbol{R}; \boldsymbol{W}, \sigma_p^2 \boldsymbol{I})$$

$$\log p(\boldsymbol{R}) = \log \left(\frac{1}{\sqrt{(2\pi\sigma_p^2)^d}} \exp\left(-\frac{1}{2\sigma_p^2} ||\boldsymbol{R} - \boldsymbol{W}||_2^2 \right) \right)$$

$$= C_1 ||\boldsymbol{R} - \boldsymbol{W}||_2^2 + C_2,$$

Using the Bayes rule, this objective can be further decomposed into prior and likelihood, which is

$$\log p(\boldsymbol{R}|G) = \log p(\boldsymbol{R}) + \log p(G|\boldsymbol{R}) + C_0, \quad (8)$$

where C_0 is a constant. For the first term, we assume that the prototypes $\boldsymbol{R}$ follow isotropic Gaussian distribution, *i.e.* $\boldsymbol{R} \sim \mathcal{N}(\boldsymbol{R}; \boldsymbol{W}, \sigma_p^2 \boldsymbol{I})$, where the mean is set to the initial values of the prototypes, and σ_p^2 is the shared variance of the prototypes. Under this assumption, the prior knowledge constraint of the prototypes can be written as:

$$\begin{aligned}\log p(\boldsymbol{R}) &= \log\Big(\frac{1}{\sqrt{(2\pi\sigma_p^2)^d}} \exp\big(-\frac{1}{2\sigma_p^2}||\boldsymbol{R}-\boldsymbol{W}||_2^2\big)\Big) \\ &= C_1||\boldsymbol{R}-\boldsymbol{W}||_2^2 + C_2,\end{aligned} \quad (9)$$

where C_1 and C_2 are constants.

For the second term, we transform it into a self-supervised objective. Inspired by the mixture models (McLachlan & Basford, 1988; Reynolds et al., 2009), we have:

$$\begin{aligned}\log p(G|\boldsymbol{R}) = \log p(G|\{\boldsymbol{r}_i\}_{i=1}^C) &= \log \sum_{i=1}^C q(G,i) p(G|\boldsymbol{r}_i) \\ &\geq \sum_{i=1}^C q(G,i) \log p(G|\boldsymbol{r}_i)\end{aligned} \quad (10)$$

where $q(G,i)$ denotes the assignment of graph G belonging to prototype $\boldsymbol{r}_i$, and the inequality comes from Jensen's inequality. Thus, we have a lower bound for the likelihood. For the estimation of $q(G,i)$, a common approach is to compute the similarity between the representation of the graph $\boldsymbol{z}^G$ and the prototypes $\boldsymbol{r}_i$. Denoting $\boldsymbol{q}^G = [q(G,1), q(G,2), \cdots q(G,C)]^T$, this simple objective can be written as

$$\underset{\boldsymbol{q}^G}{\text{argmin}} \langle \boldsymbol{q}^G, \boldsymbol{R}^T \boldsymbol{z}^G \rangle, \text{ s.t.} \langle \boldsymbol{q}^G, \boldsymbol{1}_C \rangle = 1, \quad (11)$$

where $\langle \cdot, \cdot \rangle$ denotes inner product measuring the similarity, and $\boldsymbol{1}_C \in \mathbb{R}^C$ is an all-one vector. However, this simple objective can result in unbalanced estimation, where the number of assignments to some classes is substantially larger than others. To solve this problem, another constraint is needed. Concretely, given a set of graphs $G_1, G_2, \cdots, G_N$, where N is the number of graphs, we hope to achieve a balanced assignment, which can be written as

$$\underset{\boldsymbol{Q}}{\text{argmin}}\ \text{tr}(\boldsymbol{Q}^T \boldsymbol{R}^T \boldsymbol{Z}), \text{ s.t.} \boldsymbol{Q}\boldsymbol{1}_N = \frac{N}{C}\boldsymbol{1}_C,\ \boldsymbol{Q}^T\boldsymbol{1}_C = \boldsymbol{1}_N, \quad (12)$$

where $\boldsymbol{Q} = [\boldsymbol{q}^{G_1}, \boldsymbol{q}^{G_2}, \cdots, \boldsymbol{q}^{G_N}] \in \mathbb{R}^{C\times N}$ is the stacked assignments, and $\boldsymbol{Z} = [\boldsymbol{z}^{G_1}, \boldsymbol{z}^{G_2}, \cdots, \boldsymbol{z}^{G_N}] \in \mathbb{R}^{d\times N}$ is the stacked graph representations. Solving this problem, however, requires solving a large linear program (Genevay et al., 2019), and a common workaround is to add entropy regularization that encourages the diversity of assignment. A Lagrangian objective can be written as follows

$$\begin{aligned}\mathcal{E} = \text{tr}(\boldsymbol{Q}^T \boldsymbol{R}^T \boldsymbol{Z}) + \epsilon \mathcal{H}(\boldsymbol{Q}) + \langle \boldsymbol{a}, \boldsymbol{Q}\boldsymbol{1}_N - \frac{N}{C}\boldsymbol{1}_C \rangle \\ + \langle \boldsymbol{b}, \boldsymbol{Q}^T\boldsymbol{1}_C - \boldsymbol{1}_N \rangle,\end{aligned} \quad (13)$$

where $\mathcal{E}$ is the objective to be minimized, ϵ is the temperature parameters controlling the degree of regularization, $\boldsymbol{a}$ and $\boldsymbol{b}$ are Lagrangian multipliers, $\mathcal{H}(\boldsymbol{Q}) = -\sum_{i,j} \boldsymbol{Q}_{i,j}(\log \boldsymbol{Q}_{i,j} - 1)$ is the entropy regularization. We take the derivative with respect to $\boldsymbol{Q}_{i,j}$, which gives the following result:

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{Q}_{i,j}} = [\boldsymbol{R}^T \boldsymbol{Z}]_{i,j} - \epsilon \log \boldsymbol{Q}_{i,j} + \boldsymbol{a}_i + \boldsymbol{b}_j. \quad (14)$$

Therefore, the closed-form solution can be written as:

$$\boldsymbol{Q}^* = \text{diag}\Big(\exp\Big(\frac{\boldsymbol{a}}{\epsilon}\Big)\Big) \exp\Big(\frac{\boldsymbol{R}^T\boldsymbol{Z}}{\epsilon}\Big) \text{diag}\Big(\exp\Big(\frac{\boldsymbol{b}}{\epsilon}\Big)\Big). \quad (15)$$

In practice, we utilize the iterative Sinkhorn-Knopp algorithm (Asano et al., 2019; Caron et al., 2020; Zheng et al., 2021) to solve this problem (more details in Appendix B). Specifically, repeated row normalization and column normalization are conducted on $\exp\left(\frac{\boldsymbol{R}^T\boldsymbol{Z}}{\epsilon}\right)$, and the algorithm converges quickly. Previous studies (Genevay et al., 2019; Luo et al., 2023) indicate satisfactory performance within 3 iterations, and the soft assignment of the entropy is beneficial.

Then, we provide an estimation of $p(G|\boldsymbol{r}_i)$. Although this probability takes the form of generative modeling, actually training a generative model can be costly, and when facing insufficient data, it can be inaccurate. Therefore, we estimate this term in the latent space using the feature extractor f_θ with isotropic Gaussian distribution, which is $p(G|\boldsymbol{r}_i) \propto \exp\left(-\frac{1}{2\sigma_z^2}||f_\theta(G) - \boldsymbol{r}_i||_2^2\right)$, where σ_z^2 is the shared variance of the latent features. Combining the two terms, the loss function can be written as

$$\mathcal{L}_{RPS} = \sum_{i=1}^C q(G,i)||f_\theta(G) - \boldsymbol{r}_i||_2^2 + \alpha_1||\boldsymbol{R}-\boldsymbol{W}||_2^2, \quad (16)$$

where $q(G,i)$ is obtained via the Sinkhorn-Knopp iterations within a batch of graphs, and α_1 is the hyperparameter balancing the two terms. The first term serves as a self-training objective utilizing the posterior information from the unlabeled data, whereas the second term serves as regularization from the prior knowledge of the unknown training graphs. As the prototypes $\boldsymbol{r}_i$ (with their matrix form $\boldsymbol{R}$) are used to provide supervision signals in the self-training and regularized by the prior knowledge, we call this method regularized prototype supervision.


```
if args.update_sinkhorn:
    sim_mat = F.normalize(x_emb, dim=1) @ F.normalize(prototypes, dim=1).T  # B x C
    sh_prob = sinkhorn(sim_mat.detach(), eps=args.sinkhorn_eps, n_iters=5)  # B x C
```

$$\mathcal{L} = \mathcal{L}_{RPS} + \alpha_2 \mathcal{L}_{MI},$$

$$\mathcal{L}_{MI} = -I_{\theta,\phi}(G; \tilde{G})$$

Method

Algorithm 1 Optimization Algorithm of ASSESS

Input: The well-trained GNN-based model on the training graphs $\mathcal{M}^{(0)}(\cdot)$, test graphs $\mathcal{D}^{te}$,
Output: GNN-based model after T epochs $\mathcal{M}^{(T)}(\cdot)$

Initialize the prototypes $\boldsymbol{R} \leftarrow \boldsymbol{W}$;
Initialize the reliable test graph set $\mathcal{D}^{(0)}_{ASBS} = \mathcal{D}^{conf}$ using Eq. 1;
for $i = 1, 2, \cdots, T$ **do**
  for each batch **do**
    Sample a mini-batch of target graphs;
    Calculate $\mathcal{L}_{RPS}$ using Eq. 16 over $\mathcal{D}^{(i)}_{ASBS}$;
    Calculate $\mathcal{L}_{MI}$ using Eq. 3 over $\mathcal{D}^{te}$;
    Calculate the loss objective using Eq. 17;
    Update parameters of $\mathcal{M}^{(i)}$ through back-propagation;
  end for
  Update the threshold correction function using Eq. 5;
  Obtain reliable graphs $\mathcal{D}^{(i+1)}_{ASBS}$ using Eq. 6;
end for

Experiment

Table 2. The classification accuracy (in %, training → test) on PROTEINS (P0, P1, P2) and NCI1 (N0, N1, N2).

Methods	PROTEINS							NCI1						
	P0→P1	P0→P2	P1→P0	P1→P2	P2→P0	P2→P1	AVG	N0→N1	N0→N2	N1→N0	N1→N2	N2→N0	N2→N1	AVG
GCN	57.0±4.8	41.9±0.7	64.8±2.0	54.7±3.3	62.5±1.3	61.7±3.5	57.1±2.6	70.1±3.1	54.8±1.3	52.6±2.0	56.3±1.9	50.1±3.3	63.0±1.5	57.8±2.2
GraphSAGE	53.9±5.3	41.5±0.5	70.3±3.2	57.3±1.9	69.0±0.9	62.1±2.1	59.0±2.3	69.2±1.0	55.8±0.8	51.2±1.2	59.5±2.7	48.8±0.8	61.6±3.7	57.7±1.7
GIN	58.9±1.1	41.7±0.6	64.2±1.9	53.1±2.2	59.2±0.7	68.2±2.4	57.6±1.5	73.0±2.5	56.9±1.5	55.6±4.7	59.1±2.2	50.3±0.9	64.5±1.9	59.9±2.3
GAT	63.2±4.3	41.5±0.5	77.1±2.9	63.8±4.1	64.4±2.9	61.0±1.8	61.8±2.8	70.5±6.9	55.6±0.4	51.9±1.4	57.4±1.9	50.7±1.2	63.9±2.9	58.3±2.5
MeanTeacher	72.0±7.2	65.3±5.2	70.5±2.8	66.5±7.3	73.3±5.1	67.2±2.6	69.1±5.0	71.5±8.6	61.9±3.7	64.7±10.5	75.4±2.5	59.4±12.3	66.1±1.5	66.5±6.5
GraphCL	79.2±6.0	73.7±6.4	75.2±5.2	73.9±4.1	73.5±5.1	77.0±7.4	75.4±5.7	76.2±3.3	67.6±1.3	69.0±1.1	77.3±1.5	66.2±3.2	73.1±3.2	71.6±2.3
SHOT	68.2±5.4	62.9±4.2	71.6±4.2	65.9±7.0	67.8±6.0	68.8±7.0	67.5±5.6	64.4±4.4	56.6±2.6	73.9±1.1	65.5±3.4	69.6±2.7	64.0±2.8	65.7±2.8
TAST	68.9±7.5	41.9±1.0	75.1±2.2	62.1±2.6	63.8±2.5	61.9±1.6	62.3±2.9	72.2±5.1	56.5±0.8	55.2±2.0	57.7±2.3	48.9±0.7	66.0±3.7	59.4±2.4
RNA	61.8±9.1	65.8±4.1	79.1±2.3	71.1±9.1	74.2±3.1	69.3±1.9	70.2±4.9	72.0±1.8	61.3±4.7	76.4±3.7	72.4±1.1	65.7±4.4	74.9±0.5	70.5±2.7
Ours	81.9±3.7	74.7±4.1	86.7±1.7	77.3±2.8	82.4±4.0	70.4±6.6	**78.9±3.8**	79.6±2.1	69.1±2.5	81.9±2.0	78.5±1.1	73.4±2.0	70.3±1.1	**75.5±1.8**

Table 3. The classification results (in %, training → test) on IMDB-BINARY (I0, I1, I2).

Methods	I0→I1	I0→I2	I1→I0	I1→I2	I2→I0	I2→I1	AVG
GCN	78.5±2.0	66.0±2.9	60.6±1.8	62.4±1.7	55.2±1.3	57.0±3.3	63.3±2.2
GraphSAGE	78.2±1.8	60.6±4.1	58.5±3.2	63.9±1.5	56.7±0.9	56.7±2.7	62.4±2.4
GIN	75.8±1.7	66.6±2.2	58.8±5.6	64.5±1.7	54.3±0.7	56.7±2.1	62.8±2.4
GAT	77.3±1.1	64.5±1.7	63.6±0.7	62.1±2.0	54.9±2.2	58.8±0.7	63.5±1.4
MeanTeacher	70.1±6.8	65.2±2.5	63.2±4.3	70.1±2.4	59.7±2.4	61.2±4.2	64.9±3.8
GraphCL	66.2±3.1	70.6±3.1	61.7±1.9	70.1±2.4	61.7±4.9	46.3±4.4	62.8±3.3
SHOT	68.9±5.1	66.6±4.5	65.7±3.4	67.8±6.8	61.5±7.3	60.6±9.4	65.2±6.1
TAST	80.3±2.2	65.1±3.8	58.2±2.1	64.2±1.9	57.6±0.7	58.2±2.5	63.9±2.2
RNA	64.2±3.2	67.2±4.9	59.7±3.2	69.2±2.5	61.7±3.5	68.7±3.2	65.1±3.4
Ours	82.1±3.7	66.7±3.7	63.7±1.9	63.7±1.9	69.7±3.1	61.2±3.2	**67.8±2.9**

Three types:

1. GNN.
2. Un/semi-supervised.
3. TTA

The latest baseline is RNA (Luo et al., 2024)

The unsupervised / semisupervised learning methods are not the optimal solution. Lack of consideration of test-time distribution shifts.

TTA methods designed for Euclidean data are also not very satisfactory on graphs.

Experiment

Table 4. Ablation studies on three datasets, *i.e.* FRANKENSTEIN (FRAN), Mutangenicity (MUTA), and NCI1. Average accuracy over six settings (in %) is reported.

Experiments	FRAN	MUTA	NCI1	AVG
w/o ASBS	59.8±2.0	72.4±1.3	73.9±1.4	68.7
w/o RPS-a	59.4±1.5	71.3±1.1	74.0±1.7	68.2
w/o RPS-b	59.4±1.2	70.6±1.5	74.1±1.8	68.0
Full Model	60.3±1.9	73.5±1.5	75.5±1.8	69.8

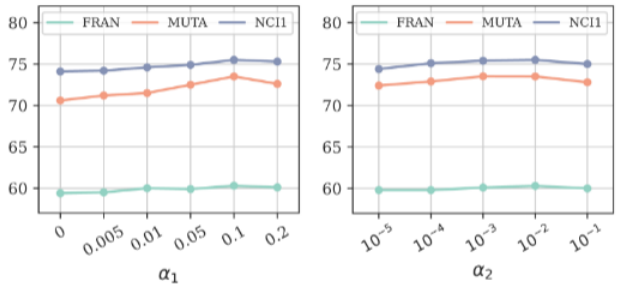


Figure 2. The model's sensitivity to hyperparameters (*i.e.* α_1 and α_2). The experiments are performed on FRANKENSTEIN (FRAN), Mutangenicity (MUTA), and NCI1. Average accuracy (in %) are reported over six settings.

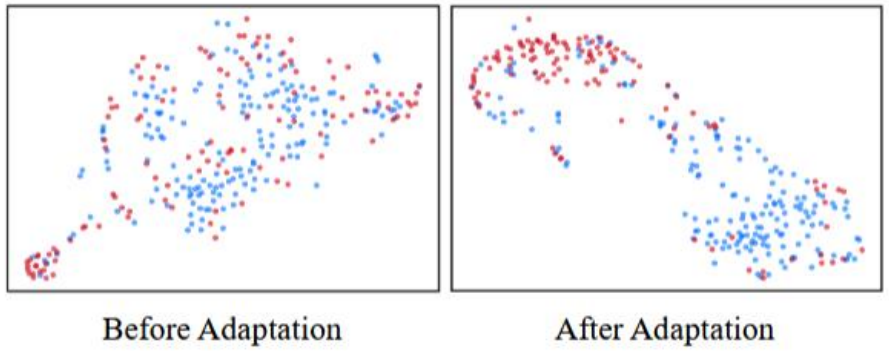


Figure 3. Visualization of learned representations on the PROTEINS dataset, before and after adaptation in the P1→P0 setting.

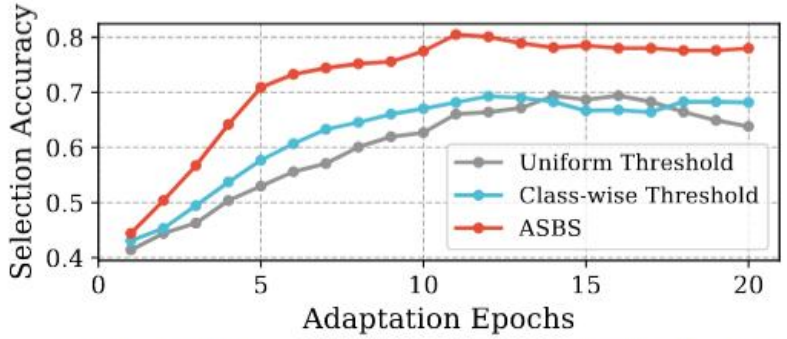


Figure 4. The selection accuracy using uniform threshold, class-wise threshold, and the proposed ASBS on the PROTEINS dataset.

Thank you