Avoiding Structural Pitfalls: Self-Supervised Low-Rank Feature Tuning for Graph Test-Time Adaptation

Anonymous authors
Paper under double-blind review

Abstract

Pre-trained graph neural networks (GNNs) have demonstrated significant success in leveraging large-scale graph data to learn transferable representations. However, their performance often degrades under distribution shifts, particularly in real-world scenarios where labels of test data are unavailable. To address this challenge, we propose **G**raph **O**ptimization via **A**ugmented **T**ransformations (GOAT), a novel self-supervised test-time tuning paradigm that adapts pre-trained GNNs to distribution-shifted test data by focusing exclusively on node feature transformations. By avoiding complex and often suboptimal graph structure transformations, GOAT overcomes the limitations of existing data-centric methods. To further address the issue of transformation collapse, where feature transformations converge to trivial solutions, we introduce a parameter-efficient low-rank adapter that generates diverse transformations tailored to individual input graphs. This design not only enhances adaptation performance but also improves interpretability by avoiding modifications to the graph structure. Through extensive experiments on six real-world datasets with diverse distribution shifts, we demonstrate that GOAT achieves consistent performance improvements across different pre-trained GNN backbones, outperforming state-of-the-art test-time adaptation methods.

1 Introduction

Graph pre-training has become a powerful technique for capturing and preserving information from large-scale upstream graph data (Kipf & Welling, 2022; Hamilton et al., 2017; Veličković et al., 2018), enabling graph neural networks (GNNs) to learn rich representations that can be effectively transferred to a variety of downstream graph tasks. However, the performance of pre-trained GNNs is often hindered by distribution shifts (Yehudai et al., 2021; Li et al., 2022a; Song & Wang, 2022; Zhu et al., 2021), particularly in real-world scenarios where the labels of the test data are unavailable. This presents a significant obstacle to the practical deployment of GNNs, as their performance can degrade substantially under such conditions. For example, Table 1 illustrates that the performance of a pre-trained GNN deteriorates as the distribution of the test data changes over time.

Adapting pre-trained GNNs to such unlabeled and distribution-shifted test scenarios is therefore a crucial yet under-addressed problem. Several methods have been proposed to deal with distribution shift at *training time*, including invariant risk minimization (Arjovsky et al., 2019; Wu et al., 2023), domain-invariant learning (Muandet et al., 2013; Li et al., 2022b), and invariant representation learning (Wu et al., 2022; Chen et al., 2022b). These approaches attempt to learn features that remain stable across environments, assuming access to labeled data in the target domain during training. Unfortunately, in many practical settings, test-time data is both unlabeled and unavailable at training time, rendering these methods inapplicable for adapting pre-trained models at test time.

Test-time adaptation (TTA) offers a promising alternative. Model-centric TTA methods, such as updating classifier heads (Wang et al., 2022) or fine-tuning the full model (Zhang et al., 2024; Wang et al., 2021), aim to leverage the generalization capabilities of pre-trained models. However, these approaches often struggle to adapt effectively to unseen or shifted distributions, as they heavily depend on parameters and statistics

Dataset	OGB-ArXiv			Elliptic		
Description	Open-world dataset of academic papers, the graph evolves as new papers are cited.			A dataset of transactions labeled as licit or illicit, influenced by market conditions.		
Split	**Year Slice**	**Accuracy**	**Degrade**	**Time Slice**	**F1 score**	**Degrade**
Train	before 2011 - 2011	47.88%		7^{th} - 11^{th}	90.12%	
Val	2011 - 2014	44.46%	-9.96%	12^{th} - 17^{th}	78.75%	-39.17%
Test	2014 - 2020	38.92%		17^{th} - 49^{th}	50.95%	

Table 1: The showcase indicates a significant decrease in the node classification performance of the pre-trained GCN on the OGB-ArXiv (Hu et al., 2020) and Elliptic (Pareja et al., 2020) datasets in an OOD setting where graph data is generated from different time intervals. For OGB-ArXiv, the year ranges from before 2011 to 2020; for Elliptic, from the 7^{th} snapshot (when the dark market crackdown occurred) to the 49^{th} snapshot. Performance degrades noticeably during validation and testing as time progresses.

learned from the training data (Hendrycks & Dietterich, 2019; Arjovsky et al., 2019). Moreover, fine-tuning the entire model can be computationally expensive, making it impractical in many real-world resource-constrained scenarios. In contrast to model-centric approaches, data-centric TTA methods (Jin et al., 2023; Chen et al., 2022a; Zhang et al., 2024) shift the focus from updating model parameters to refining the test graph itself. By transforming node features and graph structure, these methods aim to better align the test data with the representation space of the pre-trained model. This strategy has shown considerable promise in improving model performance under distribution shifts.

Despite their effectiveness, data-centric methods typically rely on simultaneous transformations of both node features and graph structures, as adapting node features alone often leads to suboptimal performance. However, this requirement introduces two key challenges. First, identifying and optimizing appropriate graph structure transformations is inherently difficult due to the vast and continuous search space. Inaccurate structure modifications can result in poorly refined test graphs, ultimately degrading adaptation performance. Second, altering the graph structure reduces the interpretability of the adaptation process by obscuring the relationship between the original and transformed graphs, making it difficult to trace how specific structural changes influence model predictions.

While one might consider avoiding structure transformations by focusing exclusively on node features, this approach introduces its own difficulties. In particular, it risks both reduced performance and transformation collapse, where the transformation function becomes ineffective, often converging to a trivial or identity mapping that fails to facilitate meaningful adaptation.

Present work. To address these limitations and overcome the associated challenges, we propose a novel self-supervised test-time tuning paradigm - **G**raph **O**ptimization via **A**ugmented **T**ransformations (GOAT) that enables the pre-trained GNN to dynamically adapt to unseen test distributions without requiring access to test labels, source training data, or training details. 1) To avoid modifying the test graph structure, our method GOAT focuses exclusively on node feature transformations. However, naïvely transforming node features is insufficient for effective adaptation. To overcome this, we introduce a self-supervised strategy that estimates optimal node feature transformations by leveraging a set of augmented test graphs. By measuring the embedding discrepancies between augmented graphs before and after transformation, our method learns a generalizable transformation function tailored to the test distribution. 2) To address the transformation collapse issue, we propose a low-rank adapter that generates unique transformations for different input graphs while effectively utilizing the knowledge encoded in the pre-trained GNN. This approach ensures that, during each learning iteration, the augmented graphs undergo diverse transformations by the dynamic adapter, preventing uniform transformations across all graphs and preserving the diversity of the augmented data. With the learned low-rank adapter, the model's interpretability becomes more intuitive compared to interpreting transformations of the graph structure. To summarize, our main contributions are as follows:

- Our work centers on data-centric test-time adaptation and introduces a novel self-supervised test-time tuning paradigm, GOAT, that leverages node features exclusively, bypassing the complexity of optimizing for structure transformations.

- We propose a parameter-efficient low-rank adapter to address the transformation collapse issue while enhancing model interpretability.
- Extensive experiments on real-world datasets with diverse distribution shifts demonstrate consistent performance improvements across various backbones.

2 Related Work

Distribution Shift on Graphs. Graph-structured data often exhibits distribution-shift phenomena (Song & Wang, 2022; Li et al., 2022a). To tackle this challenge, researchers have proposed methods for learning invariant representations (Wu et al., 2023; Arjovsky et al., 2019; Wu et al., 2022; Chen et al., 2022b; Li et al., 2022b; Muandet et al., 2013), generalizing pre-trained GNNs (Zhu et al., 2021; Li et al., 2022a; Song & Wang, 2022; Hu et al.; Zhao et al., 2021), detecting OOD instances (Zellinger et al., 2022; Guo et al., 2023; Huang et al., 2024). Most of these approaches often require access to multiple source domains, rely on specific model architectures and train-time paradigms, or may lead to performance degradation. For a thorough review, we refer the readers to two recent surveys (Wu et al., 2024; Liu & Ding, 2024).

Graph Test-time Adaptation. Graph test-time adaptation (GTTA) aims to adapt pre-trained models to the test distribution without requiring labeled data or modifying the model's parameters (Chen et al., 2022a). Existing proposed methods can be broadly categorized into the following two classes: data-centric and model-centric. *(1) Model-centric.* These approaches center on the learning process or the design of the graph model. SLAPS (Fatemi et al., 2021), TTT (Sun et al., 2020), TTT++ (Liu et al., 2021), and Tent (Wang et al., 2021), update the whole pre-train model or some specific layers. In addition, GraphTTA (Chen et al., 2022a) and GT3 (Wang et al., 2022) tune a new classifier head expecting better prediction. However, these approaches have limitations such as reliance on specific architectures, over-smoothing, or being preoccupied with how to select negative samples for contrastive learning. These issues direct our attention to data-centric approaches. *(2) Data-centric.* These recently emerging approaches emphasize the manipulation of input graphs. GTrans (Jin et al., 2023) both modify the adjacency matrix and node feature to empower graph representation learning. GraphCTA (Zhang et al., 2024) uses a memory bank to keep the best neighbor structure. Whereas, GTrans requires the transformation of graph structure, leading to the suboptimal structure refinement, while GraphCTA still needs to update the whole pre-trained model's parameters. Graph prompt tuning (Fang et al., 2024) inspires a succinct direction; however, it emphasizes the task discrepancy rather than the distribution shift and relies on the availability of label sets for downstream tasks. These challenges guide us toward a more universal test-time tuning paradigm.

3 Methodology

In this section, we delve into our proposed paradigm GOAT and Figure 1 provides the framework overview.

3.1 Self-supervised Graph Test-time Adaptation from Augmentations

A graph is represented by $\mathcal{G}$ with a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. The adjacency matrix derived from a graph is denoted by $\mathcal{A} \in \mathbb{R}^{N\times N}$. If $v_i, v_j \in \mathcal{V}$ and $(v_i, v_j) \in \mathcal{E}$, then A_{ij} is one, otherwise it is zero. For any input graph data $\mathcal{G} = (\mathcal{A}, \mathcal{X})$, $\mathcal{A} = \{a_{vu} \mid v, u \in \mathcal{V}\}$ is the adjacency matrix and $\mathcal{X} = \{x_v \mid v \in \mathcal{V}\}$ is the node features. Apart from these, each node in the graph has a label y_v, and the label of the whole graph can be represented as a vector $\mathcal{Y}$. In a local view, we define an induced graph $\mathcal{G}_S = (\mathcal{A}_S, \mathcal{X}_S)$, where $S \subseteq \mathcal{V}$, $\mathcal{A}_S = \{a_{vu} \mid v, u \in S\}$, and $\mathcal{X}_S = \{x_v \mid v \in S\}$. Its corresponding label is $\mathcal{Y}_S$. During train time, a GNN $f_\theta(\mathcal{G})$ parameterized by θ is optimized by the input-target pairs $(\mathcal{G}_S, \mathcal{Y}_S)$ sampled from training data $\widetilde{D}_{tr} = \{\mathcal{G}_{tr}, \mathcal{Y}_{tr}\}$, which minimizes the empirical risk:

$$\theta^\star = \arg\min_\theta \mathop{\mathbb{E}}_{(\mathcal{G}_S, \mathcal{Y}_S)} \mathcal{L}_{tr}(f_\theta(\mathcal{G}_S), \mathcal{Y}_S). \tag{1}$$

At test time, for arbitrary graph $\mathcal{G}_{te}$, when the label $\mathcal{Y}_{te}$ is available, following the data-centric paradigm (Jin et al., 2023; Zhang et al., 2024), an extra transformation of the input graph data should satisfy the

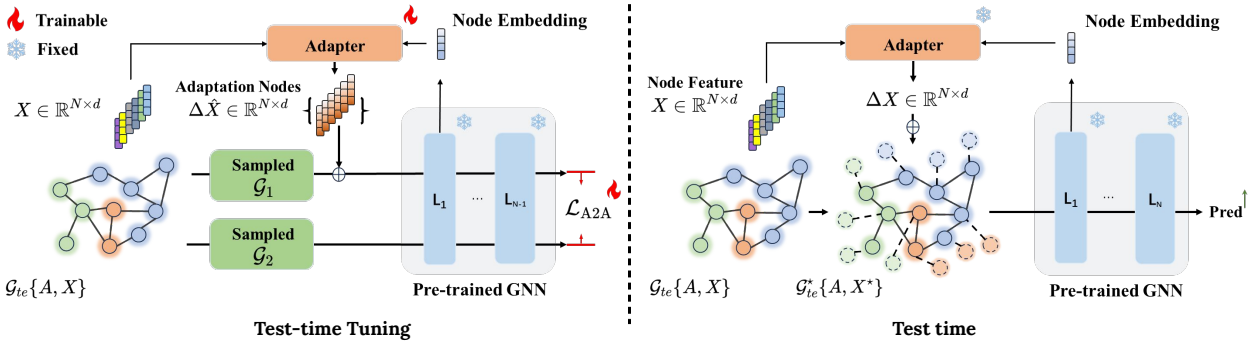


Figure 1: Overview of the proposed method GOAT under two augmented views. In the **test-time tuning**, multiple augmented views of a single test graph are generated by extracting induced subgraphs. These views are passed through the fixed pre-trained GNN, with their input node features modified by an adaptation term ΔX. A self-supervised loss $\mathcal{L}_{\text{A2A}}$ is used to train the adapter, encouraging alignment between the embeddings of one view with node feature transformation and another without. During the **test-time inference** phase, all parameters are frozen. The learned adaptation ΔX is added to the input features of the test graph and passed through the pre-trained GNN, resulting in improved prediction performance.

following objective:

$$\psi^{\star} = \arg\min_{\psi} \mathop{\mathbb{E}_{\mathcal{Y}}}_{(\mathcal{G}_{te}, \mathcal{Y}_{te})} \mathcal{L}_{te}(f_{\theta^{\star}}(g_{\psi}(\mathcal{G}_{te})), \mathcal{Y}_{te}), \tag{2}$$

denoted that for parameter efficiency, the parameters $\theta^{\star}$ of the pre-trained GNN $f(\mathcal{G})$ is fixed during tuning the graph transformation $g(\mathcal{G}) : \mathbb{G} \to \mathbb{G}$ parameterized by ψ and $|\psi|$ should be significantly smaller than $|\theta^{\star}|$. During the prediction stage, $\mathcal{L}_{te}$ represents the likelihood loss, while in the representation learning stage, $\mathcal{L}_{te}$ can be defined as the L_2 norm. Since the pre-trained $\theta^{\star}$ are fixed at test-time and the test graph is the focus of this paper, we drop the subscript in $\mathcal{G}_{te}$ and $f_{\theta^{\star}}$ to simplify notations in the rest of the paper.

When test labels $\mathcal{Y}_{te}$ are unavailable, we reframe the original task of predicting $\mathcal{Y}_{te}$ as estimating the optimal representation $z^{\star} = f(g^{\star}_{\psi}(\mathcal{G}_{te}))$, where f is the fixed pre-trained GNN encoder, and g_{ψ} is a transformation function applied at test time. To estimate the optimal representation $z^{\star}$, we use *proxies*, a strategy akin to estimating an unknown quantity (e.g., room temperature) by averaging multiple noisy measurements. According to the law of large numbers, this can yield a reasonably accurate approximation of the ground truth. Since we do not modify the pre-trained model or access ground-truth labels, we optimize g_{ψ} using *proxy representations* $\hat{z}$ obtained from augmented versions of the test graph, denoted $\hat{\mathcal{G}}_{te}$:

$$\hat{z} = f(\hat{\mathcal{G}}_{te}). \tag{3}$$

Using these proxies, we formulate the objective to learn g_{ψ} as:

$$\psi^{\star} = \arg\min_{\psi} \mathbb{E}_{(\mathcal{G}_{te}, \hat{z})} \left\| f(g_{\psi}(\mathcal{G}_{te})) - \hat{z} \right\|^2, \tag{4}$$

where the L_2 norm is used as the loss $\mathcal{L}_{te}$ in the representation learning stage. Since we use L_2 loss, it suffices that the proxy $\hat{z}$ satisfies $\mathbb{E}_{\hat{z}}[\hat{z}] = \mathbb{E}_{z}[z^{\star}]$.

To ensure generalization, g_{ψ} must be capable of transforming both the original test graph $\mathcal{G}_{te}$ and its various augmented versions $\hat{\mathcal{G}}_{te}$. To this end, we train g_{ψ} on $\mathcal{G}_{te}$ and generate proxies from $\hat{\mathcal{G}}_{te}$. Importantly, to prevent g_{ψ} from collapsing to the identity function, the input graph passed to g_{ψ} must be different from the one that generates $\hat{z}$. Specifically, let $\widetilde{\mathcal{D}}_{aug} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{|\tau|}\}$ be a set of $|\tau|$ augmented graphs derived from $\mathcal{G}_{te}$. We train g_{ψ} to transform each $\mathcal{G}_p \in \widetilde{\mathcal{D}}_{aug}$, while selecting different graphs $\mathcal{G}_q \in \widetilde{\mathcal{D}}_{aug} \setminus \mathcal{G}_p$ as the

proxies. The optimization objective becomes (proofs can be found in Appendix A):

$$\mathcal{L}_s = \arg\min_{\psi} \mathbb{E}_{\widetilde{\mathcal{D}}_{aug}} \left[\frac{1}{\binom{|\tau|}{2}} \sum_{p=1}^{|\tau|} \sum_{\substack{q=1 \\ q \neq p}}^{|\tau|} \left\| f(g_\psi(\mathcal{G}_p)) - f(\mathcal{G}_q) \right\|^2 \right], \tag{5}$$

subject to the constraint:

$$\mathbb{E}_\nu \left[f(g_\psi(\mathcal{G}_i)) - f(\mathcal{G}_i) \right] = 0, \quad \forall \mathcal{G}_i \in \widetilde{\mathcal{D}}_{aug}. \tag{6}$$

To efficiently construct $\widetilde{\mathcal{D}}_{aug}$, we apply a subgraph sampling strategy as in (Rong et al., 2020; Hamilton et al., 2017) to generate sufficient graph augmentations. Note that the augmentations here are applied exclusively to node features. The loss $\mathcal{L}_s(\mathcal{G}_p, \mathcal{G}_q)$ is symmetric with respect to its inputs, swapping $\mathcal{G}_p$ and $\mathcal{G}_q$ leaves the loss unchanged. This symmetry ensures that the adaptation process is invariant to the input order, thereby enhancing the robustness of the data-centric optimization approach.

3.2 Node Feature Transformation with Low-rank Cross-attention Adapter

During optimizing the transformation g_ψ with $\mathcal{L}_s$ in Eq.(5), collapse mapping occurs when each augmented graph receives the same transformation solution. Therefore, we design a low-rank cross-attention adapter to ensure that distinct inputs $\mathcal{G}$ produce unique outputs $g_\psi(\mathcal{G})$, thereby preventing all augmented graphs from being mapped to the same point in the pre-trained GNN's representation space through a data-centric perspective.

For an arbitrary graph $\mathcal{G} = (\mathcal{A}, \mathcal{X})$, where $\mathcal{A} \in \mathbb{R}^{N \times N}$ represents the adjacency matrix encoding the edge relationships among N nodes, and $\mathcal{X} \in \mathbb{R}^{N \times d}$ denotes the node feature matrix with d-dimensional features for each node, data-centric graph adaptation assumes the existence of a transformed graph $\mathcal{G}^\star = g_{\psi^\star}(\mathcal{G})$, which ensures that the graph achieves the best possible performance when applied to the pre-trained GNN $f(\cdot)$, by bridging the gap between the test graph and the knowledge learned by $f(\cdot)$. Specifically, the graph transformation is defined as:

$$g(\mathcal{G}) = (\mathcal{A}, \mathcal{X} \oplus \Delta X), \tag{7}$$

where $\Delta X \in \mathbb{R}^{N \times d}$ denotes continuous learnable parameters with the same shape as the node features. Here, $\oplus$ denotes element-wise addition.

However, simply adding ΔX cannot solve the collapse mapping issue. To better solve this issue and integrate global and local information of test graphs into ΔX while fully leveraging the knowledge learned by the pre-trained GNN, we design the following adapter incorporating the pre-trained GNN to generate ΔX dynamically.

In GNNs, learning of node representations typically relies on the aggregation of neighboring nodes. Given a graph $\mathcal{G} = (\mathcal{A}, \mathcal{X})$, an L-layer GNN can be represented as:

$$H^{(l)} = \sigma(AGG^{(l)}(\mathcal{A}, H^{(l-1)})), \quad l = 1, 2, \ldots, L, \tag{8}$$

where $H^{(l)} \in \mathbb{R}^{N \times d^{(l)}}$ represents the node representation at the l-th layer in pre-trained GNN with $H^{(0)} = \mathcal{X}$, $AGG^{(l)}(\cdot)$ is the aggregation function at the l-th layer, and σ is the activation function. In our experiments, the ReLU activation function is employed: $\sigma(x) = x^+$.

Although GNNs effectively capture local structural information through neighborhood aggregation, they face two key limitations. First, they primarily focus on local interactions and lack mechanisms to model global structural context. Second, modeling long-range dependencies requires stacking multiple layers, which significantly increases computational cost and often leads to oversmoothing. To address these challenges, we propose the Low-Rank Adapter (LRA), a lightweight module designed to incorporate global context efficiently via attention over a compact set of virtual nodes.

Specifically, the LRA operates on the k-th output of the first k layers of the GNN with the original node features to obtain an k-hop neighbor-aware node representation $H^{(k)} \in \mathbb{R}^{N \times d^{(k)}}$. Then we introduce Low-Rank projection : $\mathbb{R}^{N \times d'} \to \mathbb{R}^{|n| \times d'}$ (usually $|n| \ll N$), which is a low-rank projection function that projects nodes into a latent space represented by few virtual nodes, striving for maximal rank density and minimal computational overhead, where d' is the attention dimension in LRA. Let $E, F \in \mathbb{R}^{|n| \times N}$ denote the key and value's low-rank transition matrices of the virtual nodes. The core of LRA is a self-attention mechanism, which enables each node to attend to a globally-aware summary of the graph via virtual nodes. The computation process of LRA is:

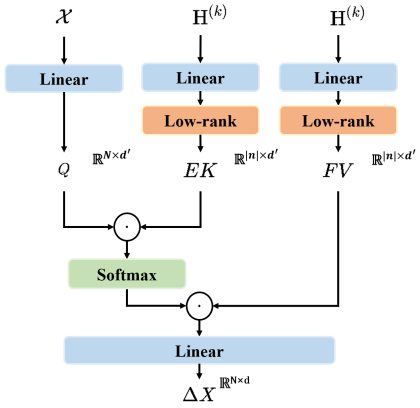


Figure 2: Structure overview of Low-Rank Adapter.

$$\Delta X = (\text{Softmax}(\frac{Q(EK)^T}{\sqrt{d'}}) \cdot (FV))W_X, \tag{9}$$

where $W_X \in \mathbb{R}^{d' \times d}$ is a learnable output projection matrix, and Q, K, V are obtained via:

$$Q = \mathcal{X}W_Q, \quad K = H^{(k)}W_K, \quad V = H^{(k)}W_V. \tag{10}$$

Here, $W_Q \in \mathbb{R}^{d \times d'}$, $W_K \in \mathbb{R}^{d^{(k)} \times d'}$, and $W_V \in \mathbb{R}^{d^{(k)} \times d'}$ are learnable weight matrices. By introducing the low-rank transition matrices E and F, LRA can compute long-range dependencies between nodes with linear time complexity and generate $\Delta X \in \mathbb{R}^{N \times d}$ that contain local-global contextual information. The time complexity of the LRA, assuming $k = 1$, is $O(Nd')$.

With LRA, both interpretability and robustness are improved. The output ΔX can be directly analyzed, making it more interpretable than structural graph transformations. Additionally, in the test-time adaptation setting, LRA helps prevent transformation collapse: when applied to different graphs $g(\mathcal{G}_q)$ and $g(\mathcal{G}_p)$, their outputs are less likely to degenerate to the same representation, thus maintaining representational diversity.

3.3 Optimization

The constraint in Eq.(5) requires the same input graph for both $f(g_\psi(\cdot))$ and $f(\cdot)$, which can easily result in the mapping collapse issue. As transformation $g(\mathcal{G})$ can be formulated as $(\mathcal{A}, \mathcal{X} + \Delta X)$, the constraint is equivalent to:

$$\begin{aligned} \arg\min_{\Delta X} \mathbb{E}_{\mathcal{V}} \| f(\mathcal{A}, \mathcal{X} + \Delta X) - f(\mathcal{A}, \mathcal{X}) - f(\mathcal{A}, \Delta X) \|^2 \\ \text{s.t.} \quad \mathbb{E}_{\mathcal{V}}[f(\mathcal{A}, \Delta X)] = 0, \end{aligned} \tag{11}$$

where $f(\mathcal{A}, \Delta X)$ serves as a regularization. We define $g_\psi(f(\mathcal{G})) = f(\mathcal{A}, \mathcal{X}) + f(\mathcal{A}, \Delta X)$ as the transformation on node embeddings. In this form, it can be seen as encouraging g_ψ to learn an isomorphic mapping with f, ensuring that the adapter's transformations are consistent with those of the GNN, i.e., $g_\psi \circ f \approx f \circ g_\psi$. This consistency facilitates the rapid optimization and performance consistency of the transformation g_ψ across different designs of f. With further relaxing the constraint in Eq.(11), we derive the following consistency loss and a regularization loss. Optimizing the two loss functions jointly can achieve the same effect as optimizing the constraints in Eq.(6):

$$\begin{aligned} \mathcal{L}_c &= \arg\min_{\psi} \mathop{\mathbb{E}_{\mathcal{V}}}_{1 \le p \le |\tau|} [\frac{1}{|\tau|} \sum_{|\tau|} \| f(g_\psi(\mathcal{G}_p)) - g_\psi(f(\mathcal{G}_p)) \|^2] \\ \mathcal{L}_R &= \arg\min_{\Delta X} \mathop{\mathbb{E}_{\mathcal{V}}}_{1 \le p \le |\tau|} [\frac{1}{|\tau|} \sum_{|\tau|} \| f(\mathcal{A}_p, \Delta X_p) \|^2], \end{aligned} \tag{12}$$

where $\mathcal{A}_p$ is the adjacent matrix of $\mathcal{G}_p \in \widetilde{D}_{aug}$, ΔX_p is generated from LRA with the input of $\mathcal{G}_p$.

Table 2: Average classification performance (%) on the test graphs. The best performance on each dataset with a specific backbone is indicated in bold, the second-best method is underlined, and Avg. indicates the average ranking of the same method compared to others on all six datasets under the same backbone. OOM indicates an out-of-memory error on 24 GB GPU memory. $^{\uparrow}$/* indicates that GOAT outperforms ERM at the confidence level 0.1/0.05 from the paired t-test.

	Datasets	Artificial Transformation		Temporal Evolution		Cross-Domain		
Backbone	Method	Amz-Photo	Cora	Elliptic	OGB-Arxiv	Twitch-E	FB-100	Avg.
GCN	ERM	92.78±1.34	93.92±0.64	54.13±1.18	36.89±0.67	56.84±1.13	53.95±0.77	4.7
	EERM	94.24±0.40	87.36±0.86	53.15±0.01	OOM	57.25±0.42	54.03±0.80	5.2
	Tent	93.84±1.53	91.64±2.37	46.72±0.06	39.34±2.76	60.01±0.95	54.11±1.50	4.2
	GCTA	91.43±1.74	93.13±2.02	55.82±3.50	37.27±3.46	60.10±0.95	54.11±1.49	3.7
	GTrans	94.32±1.34	94.76±1.94	55.07±3.61	**40.45±1.76**	**60.37±1.44**	54.17±1.23	1.8
	GOAT	$^{\uparrow}$**94.35±1.32**	**94.79±1.36**	**55.83±3.81**	*39.44±2.02	*60.15±1.30	**54.19±2.04**	**1.3**
SAGE	ERM	87.79±1.74	99.62±0.09	50.11±0.39	37.52±0.66	59.20±0.14	54.09±0.40	5.2
	EERM	95.76±0.11	99.76±0.21	60.43±0.29	OOM	60.09±0.25	OOM	5.2
	Tent	95.23±1.52	99.71±0.17	50.25±3.28	39.56±1.49	**62.05±0.22**	55.11±0.55	3.0
	GCTA	96.86±1.11	99.85±0.06	66.92±2.33	33.67±3.25	62.05±0.24	55.11±0.56	3.2
	GTrans	**97.09±1.13**	99.81±0.16	63.04±6.39	**39.74±1.14**	61.97±0.34	55.07±0.59	2.5
	GOAT	*92.54±2.51	***99.89±0.10**	***67.92±5.56**	*39.52±1.03	*61.91±0.28	***55.61±0.30**	**2.5**
GAT	ERM	94.92±2.33	95.99±0.88	49.49±1.51	37.92±0.68	57.36±0.30	48.25±1.55	3.8
	EERM	94.07±1.32	79.35±8.90	54.27±2.42	OOM	56.27±0.37	52.46±2.02	3.7
	Tent	94.96±0.87	93.54±3.50	55.29±5.22	37.41±5.20	58.93±1.50	51.22±1.99	5.3
	GCTA	94.72±1.73	**96.03±1.76**	56.00±10.11	37.8 6±2.17	58.83±1.59	51.22±1.98	3.2
	GTrans	**95.14±0.70**	95.46±1.96	**62.56±4.22**	37.52±2.68	58.84±1.49	51.27±1.91	2.5
	GOAT	94.69±0.63	94.72±2.83	*60.33±4.83	***41.13±1.96**	***58.95±1.50**	***54.20±1.10**	**2.3**
GPR	ERM	84.81±3.71	83.98±1.72	48.96±1.05	40.91±0.28	57.25±0.66	54.36±0.27	4.1
	EERM	90.87±0.52	87.16±2.39	60.08±0.03	OOM	58.75±0.29	54.21±0.42	4.0
	Tent*	-	-	-	-	-	-	-
	GCTA	91.96±0.75	92.75±2.48	66.36±3.67	44.44±0.70	59.97±0.62	54.63±0.77	2.6
	GTrans	91.97±0.84	92.70±2.46	**68.54±5.56**	**45.64±0.61**	59.84±0.89	54.48±0.66	2.4
	GOAT	***91.98±0.83**	***92.79±2.74**	*66.47±6.44	*44.78±0.69	***60.00±0.65**	***55.23±0.43**	**1.7**

$\star$ Tent cannot be applied to models that do not contain batch normalization layers.

Table 3: Summary of the experimental datasets that entail diverse distribution shifts.

Dataset	Distribution Shift	#Nodes	#Edges	#Classes	Train/Val/Test Split	Metric	Adapted From
Cora	Artificial Transformation	2,703	5,278	10	Domain-Level	Accuracy	(Yang et al., 2016)
Amazon-Photo		7,650	119,081	10	Domain-Level	Accuracy	(Shchur et al., 2018)
Elliptic	Temporal Evolution	203,769	234,355	2	Time-Aware	F1 Score	(Pareja et al., 2020)
OGB-ArXiv		169,343	1,166,243	40	Time-Aware	Accuracy	(Hu et al., 2020)
Twitch-explicit	Cross-Domain Transfer	1,912 - 9,498	31,299 - 153,138	2	Domain-Level	ROC-AUC	(Rozemberczki et al., 2021)
Facebook-100		769 - 41,536	16,656 - 1,590,655	2	Domain-Level	Accuracy	(Traud et al., 2012)

The overall loss function used for training is defined as follows:

$$\mathcal{L}_{A2A} = \alpha\lambda(\mathcal{L}_s + \mathcal{L}_c) + (1-\alpha)\mathcal{L}_R, \tag{13}$$

where α and λ are hyperparameters that control the importance of each objective. α is a hyperparameter in the range $(0, 1)$; λ is a positive hyperparameter.

4 Experiments

4.1 Adaptation on Data with Distribution Shift

Datasets. We evaluate GOAT 's performance on three types of distribution shifts across six benchmark datasets, following the experimental settings of EERM (Wu et al., 2023). The dataset statistics, along with a breakdown of these three distinct types of distribution shifts, are presented in Table 3: (1) **Artificial Transformation** for Cora (Yang et al., 2016) and Amazon-Photo (Shchur et al., 2018), where node features are replaced by synthetic features. (2) **Cross-Domain** transfers for Twitch-E (Rozemberczki et al., 2021) and FB-100 (Traud et al., 2012), involving graphs from different domains. (3) **Temporal Evolution** for Elliptic (Pareja et al., 2020) and OGB-ArXiv (Hu et al., 2020), utilizing dynamic datasets with natural evolving characteristics. The datasets are split into training/validation/test sets with ratios of: 1/1/8 for

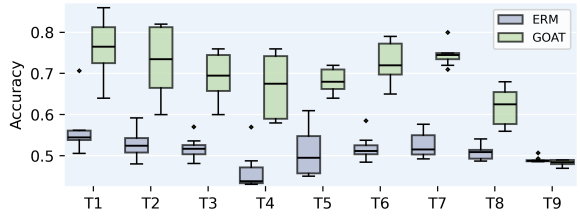


Figure 3: Results on Elliptic under OOD. GOAT improves SAGE on most test graphs.

Table 4: Efficiency comparison. GOAT is more time- and memory-efficient than EERM on large graphs and comparable to GTrans.

	Running Time (s)			GPU Memory (GB)		
	Photo	Elliptic	ArXiv	Photo	Elliptic	ArXiv
EERM	413.4	629.6	-	10.5	12.8	24+
GTrans	1.9	6.8	12.2	1.6	1.3	4.1
GOAT	5.5	0.5	0.3	1.5	1.3	5.0

Cora and Amazon-Photo; 1/1/5 for Twitch-E; 3/2/3 for FB-100; 5/5/33 for Elliptic; and 1/1/3 for OGB-ArXiv.

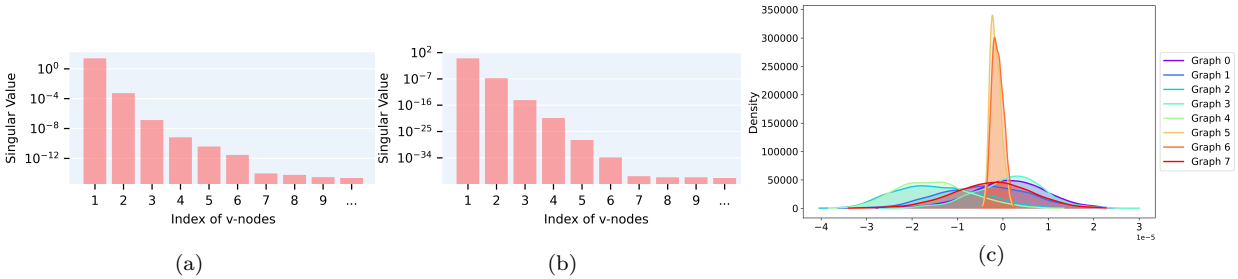
Baselines. GOAT is compared with four baselines: empirical risk minimization ERM, test-time training method Tent (Wang et al., 2021), memory-bank-based method GraphCTA(GCTA) (Zhang et al., 2024), the train-time state-of-the-art method EERM (Wu et al., 2023) which is exclusively developed for graph OOD issues, and the test-time graph transformation state-of-the-art method GTrans (Jin et al., 2023). All methods are evaluated with four popular GNN backbones: GCN (Kipf & Welling, 2022), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), and GPR (Chien et al., 2021). Their default setup follows that in EERM[1]. More implementation details of the baselines and GOAT can be found in Appendix C.1. All experiments are repeated 8 times with different random seeds. Due to page limits, additional baselines and backbones such as SR-GNN (Zhu et al., 2021), UDA-GCN (Wu et al., 2020), and GTN (Yun et al., 2019) are included in Appendix D.

Overall Comparison. Table 2 reports the averaged performance over the test graphs for each dataset as well as the average rank of each algorithm. From the table, we conduct the following observations: *(a) Overall Performance.* The proposed framework consistently achieves strong performance across the datasets: GOAT achieves average ranks of 1.3, 2.5, 2.3, and 1.7 with GCN, SAGE, GAT, and GPR, respectively, while the corresponding ranks for the best baseline GOAT are 1.8, 2.5, 2.3 and 2.4. Furthermore, in most cases, GOAT significantly improves the vanilla baseline (ERM) by a large margin. Particularly, when using SAGE as the backbone, GOAT outperforms ERM by 9.8%, 18.5%, and 3.9% on Cora, Elliptic, and OGB-ArXiv, respectively. These results demonstrate the effectiveness of GOAT in tackling diverse types of distribution shifts. *(b) Comparison to other baselines.* Both GraphCTA and EERM modify the model parameters to improve model generalization. Nonetheless, they are less effective than GOAT , as GOAT takes advantage of adapting the pre-trained GNN to the environment of test graphs. As test-time training methods, Tent and GTrans also perform well in some cases. However, Tent is ineffective for models that do not incorporate batch normalization. On the other hand, GTrans not only modifies node features but also alters edges, which can backfire if the edge modifications are not carefully chosen, potentially leading to a misrepresentation of the graph structure.

We further show the performance on each test graph on Elliptic with SAGE in Figure 3 and the results for other datasets are provided in Figure 9 in Appendix. We observe that GOAT generally improves over individual test graphs within each dataset, which validates the effectiveness of GOAT .

Efficiency Comparison. In Table 4, we compare the computational time and GPU usage on the largest graph of each dataset for our GOAT , EERM, and GTrans methods. Unlike EERM, which increases pre-training generalization through extensive environment augmentation during train time, GOAT optimizes efficiency by minimizing reliance on computationally expensive data augmentations and parameter tuning. In contrast to GTrans, which adjusts based on the proportion of edges modified on the graph, GOAT requires sampling only a minimal number of *two* augmented input graphs per training epoch. These features ensure that GOAT not only conserves GPU resources but also accelerates the adaptation process during

[1]Adjustments have only been made to the hidden dimensions of GAT to ensure consistency in the parameter count across all four backbones.



(a) (b) (c)

Figure 4: (a)(b) Visualization of the low-rank property of matrix ΔX in the LRA module of a GAT backbone trained on the two largest test graphs on OGB-ArXiv(169343 nodes) and FB-100(41554 nodes) under OOD settings. The singular values, obtained via SVD, show a rapid decay, indicating that **node embeddings can be effectively compressed into virtual nodes of the units digit**. (c) Visualization of the distribution of generated ΔX obtained after training GOAT on 8 test graphs in Cora with Gaussian KDE. The x-axis represents the sum of feature values on the nodes, and the y-axis represents the density of bias on each node within that value range. The further the mode of the distribution is from "0", the greater the degree of distribution shift.

test time, showcasing substantial efficiency improvements over both train-time methods and other test-time methods.[2]

4.2 A Node-Level Low-Rank Perspective on Adaptation

After tuning the parameters of LRA on validation sets and obtaining the optimal results through test-time tuning, we further investigate the principal components of the low-rank matrix $\Delta X \in \mathbb{R}^{N\times d}$ in the N-dimensional space as Figure 4(a)(b) shown. By performing Singular Value Decomposition (SVD), we obtained the singular values sorted in descending order and compared the major eigenvalues, showing a significant decline compared to the others. In almost all large graphs, the adaptation graph ΔX exhibits low rank along its N-th dimension. This phenomenon differs from the low-rank attention applied to the node feature across each node's dimensions. This finding further demonstrates that the input test graph can be adapted using a low-rank additive representation. To elaborate, in Eq.(7), ΔX can also take the form $\Delta X \in \mathbb{R}^{N'\times d}$ with N' nodes ($N' \ll N$), selected from a predefined node dictionary. This provides empirical evidence for setting our $|n|$ hyperparameter to a small constant that is independent of the number of nodes.

4.3 An Embedding View on Adaptation ΔX

We further visualize the t-SNE distribution of different ΔX generated by our paradigm and GTrans on different test graphs. From Figure 5, it can be observed that the ΔX obtained through our paradigm, i.e., optimized with LRA module and $\mathcal{L}_{\text{A2A}}$, forms a low-dimensional manifold $\mathcal{M}$ after being embedded by the pre-trained GNN. This implies that ΔX itself belongs to a function manifold, encouraging $g_\psi(\cdot)$ to transform each node feature *along the local tangent space of* $\mathcal{M}$ *or along the curvature directions of* $\mathcal{M}$.

Furthermore, we found that although the ΔX generated by the GTrans method still adheres to the clustering distribution of classification task labels, which, to some extent, even distorts the original topological structure learned by the GNN. A notable observation is that the performance of GTrans correlates with the degree of distortion it introduces. The distortion, in effect, allows ΔX to generate a "spurious" graph that potentially overshadows the importance of the original node features for the classification task. Upon examining the node embeddings before and after transformation by ΔX, it becomes evident that our method significantly enhances performance while maintaining a largely consistent distribution. This further demonstrates that the ΔX generated by our approach represents the degree of distributional shift, rather than conforming to the classification and clustering observed during pre-training.

[2]Detailed early-stop procedures are shown in Appendix B.

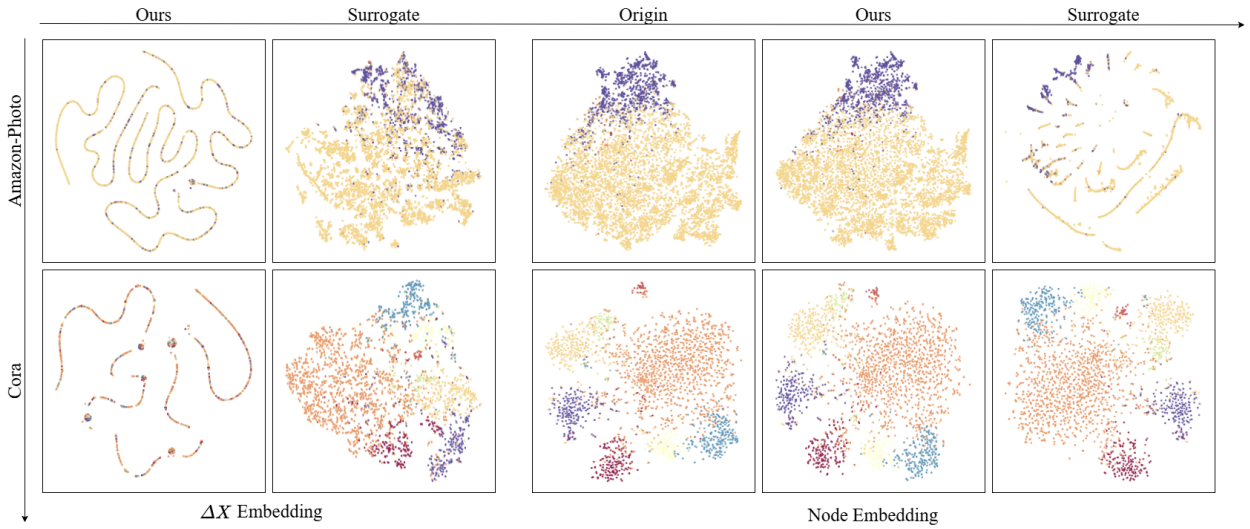


Figure 5: t-SNE visualizations of ΔX and node feature embeddings on Amazon-Photo, Cora, and Tiwtch-E datasets. The embeddings are obtained after a pre-trained GCN, using ΔX generated by our method and surrogate loss proposed by GTrans.

Table 5: Ablation study of the overall loss function $\mathcal{L}_{A2A}$ comparison on the Elliptic dataset under OOD. Two-view sampling under a test environment shows improvement in GCN average performance on test graphs with the addition of each $\mathcal{L}_{A2A}$ constraint component, demonstrating the effectiveness of each part of the loss function and the choice of the number of samples.

Configuration				Performance	
Pre-train	$\mathcal{L}_s$	$\mathcal{L}_c$	$\mathcal{L}_R$	**One sample**	**Two samples**
✓				±0.00%	±0.00%
✓	✓			-1.51%	-1.51%
✓	✓	✓		-1.62%	+4.49%
✓	✓	✓	✓	-0.18%	**+5.28%**

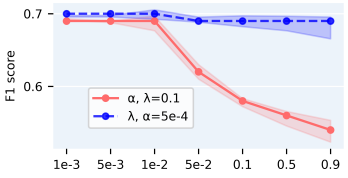


Figure 6: α, λ Parameter Study. We compared the parameter sensitivity under two commonly used settings: $\alpha = 0.0005$ and $\lambda = 0.1$

4.4 Distribution Shift Quantification

We utilize Kernel Density Estimation (KDE) to visualize the distribution of ΔX generated by our adapter obtained through GOAT on Cora and Elliptic as Figure 4(c) and Figure 7 shown, by aggregating each node's feature dimensions d in ΔX. As the initialization of the ΔX is zero, the mean and mode of the initial distribution should be 0. Due to the varying degrees of distribution shifts in different graphs, after tuning by GOAT , our adapter can effectively capture the discrepancy between the current test graph and the pre-trained GNN. Adding the generated ΔX can be seen as the mapping from the current test graph to the distribution to which the original training graph belongs. Therefore, the farther the representation deviates from the origin, the more severe distribution shift the graph has, whether observed from the perspective of the entire graph or an individual node's perspective. The distribution shifts in the time-evolving graph could be more intuitive as time flows. Furthermore, we show ΔX's distribution in other datasets and compare with the central moment discrepancy (CMD) (Zellinger et al., 2022) measurement in the Appendix.Figure 11, highlighting the interpretability of our designed adapter.

4.5 Ablation Studies and Parameter Study

Overall Loss Function $\mathcal{L}_{A2A}$. The ablation study shown in Table 5 demonstrates the effectiveness of the different components in our proposed loss function. Optimizing $\mathcal{L}_s$ alone may lead to instability and mode collapse, which empirically proves our proposed target in Eq.(5) and Eq.(12). The target $\mathcal{L}_R$ is affected by

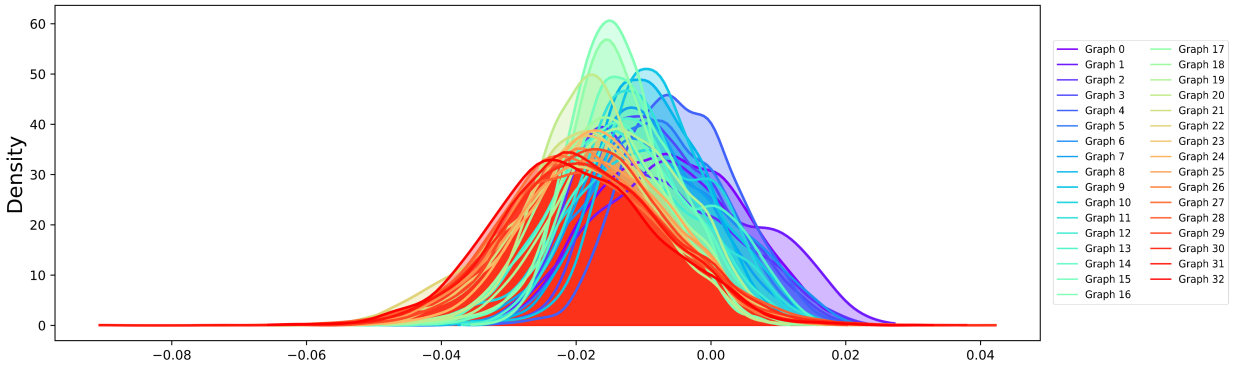


Figure 7: Visualization of the distribution of generated ΔX obtained after training GOAT on 32 test graphs in Elliptic with Gaussian KDE.

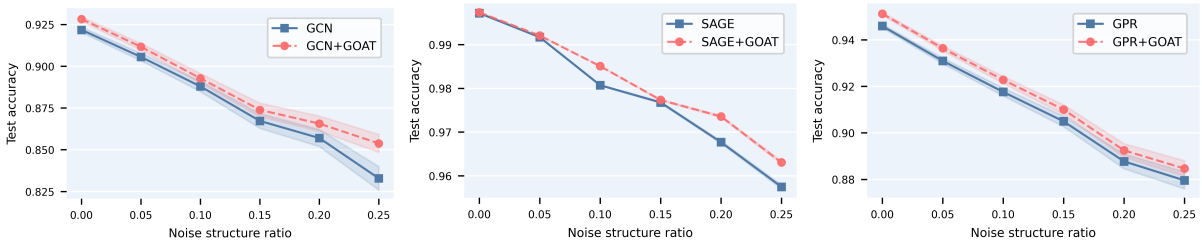


Figure 8: Performance of GCN, GraphSAGE, and GPR on Cora under structural attacks (edge addition/deletion).

whether the pre-trained GNN has sufficient generalization capability. Therefore, optimizing $\mathcal{L}_c$ alone is also sufficient to meet the requirements of Eq.(5). The best performance was achieved when all three components were jointly optimized. Additionally, sampling at least two distinct graph augmentations in $\widetilde{D}_{\text{aug}}$ is essential to prevent biased learning, which empirically supports our choice of enforcing $(q \neq p)$ in Eq.(5).

Adapter LRA. In Figure 6, we show the parameter study of λ and α in $\mathcal{L}_{A2A}$. Note that while the proportion $\frac{\lambda}{1-\alpha}$ may vary, $1-\alpha$ should remain relatively large to ensure that the constraint in Eq. (11), i.e., $\mathcal{L}_R$, is satisfied before optimizing the other objectives.

4.6 Robustness under Structure Adversarial Attack

We evaluate the robustness of GCN, GraphSAGE, and GPR on the Cora dataset (Yang et al., 2016) under structural attacks, we perturb the graph by both adding and deleting edges. For a given perturbation budget, ratio, we randomly delete a number of edges corresponding to ratio/2 of the original edge count, and randomly add an equal number of new edges. For the GCN model, our method achieves an average improvement of 2% under 25% structural attack noise, while for GPR, it delivers a stable gain of 0.5% across all noise levels. Without modifying hyperparameters (e.g., learning rate, sampling ratio), our method consistently improves backbone model performance by 0.5%–2.5% across varying attack intensities. Although our test-time paradigm consistently enhances adversarial robustness, its performance remains dependent on the base backbone's inherent generalization capability and adversarial robustness. Notably, this gain is achieved with just two samples with any subgraph sampling strategy, demonstrating computational efficiency alongside robustness.

4.7 Further Analysis

Universal Bias vs. Local-global Bias. We compare the average improvement of various non-customized additional parameter methods using our proposed $\mathcal{L}_{A2A}$ on graph classification datasets HIV (a small-scale

real-world molecular dataset adapted from MoleculeNet (Wu et al., 2018)) under the OOD setting in GOOD (Gui et al., 2022) during test time. Average results of different designs of basis are shown in Table 6.

Table 6: Bias Comparison

Method	Avg. Impr
Universal	±0.01%
Prompt dict	+0.01%
Sub-graph	+1.52%
Node-wise	+2.35%

We show its improvement of ROC-AUC(%) based on ERM with GCN. For node classification, it is evident that UPF's universal prompts (Fang et al., 2024), $\Delta X \in \mathbb{R}^{1\times d}$, across all nodes, are less customizable for classifying each node in an OOD environment, and might even learn controversial knowledge, therefore showing diminishing performance. Moreover, using a selection dictionary (Sun et al., 2023), $\Delta X \in \mathbb{R}^{k\times d}$ ($k \ll N$), also presents difficulties during test-time training. In contrast, subgraph-focused methods (Sun et al., 2022) can simultaneously capture the optimal bias more effectively, yielding relatively higher results, especially when it extends to a node-wise bias, i.e. each node's learnable bias is different, $\Delta X \in \mathbb{R}^{N\times d}$. These demonstrate that, at least in OOD node classification, a bias design that focuses on local-global context can better capture the relationships of nodes within the OOD environment. The node-wise bias method is particularly well-suited to our designed strategy, as it can better adapt to different distribution shift scenarios. This further validates the rationality of our adapter's design. Furthermore, based on the graph prompt methods (Liu et al., 2023; Yu et al., 2023), we experimented with incorporating a learnable scaling parameter that multiplies the weights of each GNN layer or node embeddings during test time. However, we found this approach difficult to apply effectively in our context.

5 Further Discussion

It is worth noting that the distribution shift issue in graph models can lead to significant risks and negative consequences in real-world applications. For instance, when GNNs are applied in financial risk control systems, distribution shifts in the input data may cause a large number of misjudgments, leading to severe economic losses or compliance issues. Similar risks exist in other high-stakes domains such as healthcare and cyber-criminal justice, where the reliability and robustness of graph-based decision-making systems under distributional changes are critical. Therefore, it is crucial to develop effective methods to detect and adapt to distribution shift scenarios in graph learning and to carefully analyze and mitigate the potential negative societal impacts. Our work aims to contribute to this important research direction.

Another illustrative example of the potential negative impact of the distribution shift issue in graph models is in the context of social network analysis for misinformation detection. GNNs have been widely adopted to identify fake news and rumors based on the propagation patterns and content features in social networks. However, the characteristics of misinformation can evolve rapidly over time, leading to distribution shifts between the training and test data. If the GNN-based misinformation detectors fail to adapt to such changes, they may miss emerging misinformation or cause false alarms, which can have severe societal consequences such as public panic, political manipulation, and erosion of trust in media. This urges the development of graph OOD detection and adaptation methods that can robustly handle the dynamic and adversarial nature of online misinformation. Our GOAT framework takes a step towards this goal by enabling test-time adaptation of GNNs to evolved data distributions.

6 Conclusion

In this work, we proposed **GOAT** , a data-centric and self-supervised test-time adaptation framework for graph neural networks that avoided the complexity and instability of structure-based transformations. By focusing exclusively on node feature optimization, GOAT enabled efficient and interpretable adaptation to distribution-shifted test graphs without requiring access to training data or test labels. To further enhance robustness, we introduced a low-rank adapter that generated diverse, graph-specific transformations and mitigated the risk of transformation collapse. Extensive experiments on six real-world datasets demonstrated that GOAT consistently improved performance across different GNN backbones, outperforming existing data-centric and model-centric test-time adaptation baselines. These results highlighted the potential of feature-only, data-centric approaches for test-time graph adaptation. In future work, we will explore a more relaxed optimization objective and inspire more discussion and novel propositions.

References

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

Guanzi Chen, Jiying Zhang, Xi Xiao, and Yang Li. Graphtta: Test time adaptation on graph neural networks. *arXiv preprint arXiv:2208.09126*, 2022a.

Yongqiang Chen, Yonggang Zhang, Yatao Bian, Han Yang, MA Kaili, Binghui Xie, Tongliang Liu, Bo Han, and James Cheng. Learning causally invariant representations for out-of-distribution generalization on graphs. *Advances in Neural Information Processing Systems*, 35:22131–22148, 2022b.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. Universal prompt tuning for graph neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 34:22667–22681, 2021.

Shurui Gui, Xiner Li, Limei Wang, and Shuiwang Ji. Good: A graph out-of-distribution benchmark. *Advances in Neural Information Processing Systems*, 35:2059–2073, 2022.

Yuxin Guo, Cheng Yang, Yuluo Chen, Jixi Liu, Chuan Shi, and Junping Du. A data-centric framework to endow graph neural networks with out-of-distribution detection ability. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 638–648, 2023.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Kexin Huang, Ying Jin, Emmanuel Candes, and Jure Leskovec. Uncertainty quantification over graph with conformalized graph neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

Wei Jin, Tong Zhao, Jiayuan Ding, Yozen Liu, Jiliang Tang, and Neil Shah. Empowering graph representation learning with test-time graph transformation. In *The Eleventh International Conference on Learning Representations*, 2023.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2022.

Haoyang Li, Xin Wang, Ziwei Zhang, and Wenwu Zhu. Ood-gnn: Out-of-distribution generalized graph neural network. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):7328–7340, 2022a.

Jinfeng Li, Weifeng Liu, Yicong Zhou, Jun Yu, Dapeng Tao, and Changsheng Xu. Domain-invariant graph for adaptive semi-supervised domain adaptation. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 18(3):1–18, 2022b.

Shuhan Liu and Kaize Ding. Beyond generalization: A survey of out-of-distribution adaptation on graphs. *arXiv preprint arXiv:2402.11153*, 2024.

Yuejiang Liu, Parth Kothari, Bastien Van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems*, 34:21808–21820, 2021.

Zemin Liu, Xingtong Yu, Yuan Fang, and Xinming Zhang. Graphprompt: Unifying pre-training and downstream tasks for graph neural networks. In *Proceedings of the ACM Web Conference 2023*, pp. 417–428, 2023.

Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. In *International conference on machine learning*, pp. 10–18. PMLR, 2013.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5363–5370, 2020.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Yu Song and Donglin Wang. Learning on graphs with out-of-distribution nodes. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1635–1645, 2022.

Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1717–1727, 2022.

Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. All in one: Multi-task prompting for graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2120–2131, 2023.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pp. 9229–9248. PMLR, 2020.

Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021.

Yiqi Wang, Chaozhuo Li, Wei Jin, Rui Li, Jianan Zhao, Jiliang Tang, and Xing Xie. Test-time training for graph neural networks. *arXiv preprint arXiv:2210.08813*, 2022.

Man Wu, Shirui Pan, Chuan Zhou, Xiaojun Chang, and Xingquan Zhu. Unsupervised domain adaptive graph convolutional networks. In *Proceedings of the web conference 2020*, pp. 1457–1467, 2020.

Man Wu, Xin Zheng, Qin Zhang, Xiao Shen, Xiong Luo, Xingquan Zhu, and Shirui Pan. Graph learning under distribution shifts: A comprehensive survey on domain adaptation, out-of-distribution, and continual learning. *arXiv preprint arXiv:2402.16374*, 2024.

Qitian Wu, Hengrui Zhang, Junchi Yan, and David Wipf. Handling distribution shifts on graphs: An invariance perspective. In *International Conference on Learning Representations*, 2023.

Yingxin Wu, Xiang Wang, An Zhang, Xiangnan He, and Tat-Seng Chua. Discovering invariant rationales for graph neural networks. In *International Conference on Learning Representations*, 2022.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pp. 11975–11986. PMLR, 2021.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

Xingtong Yu, Chang Zhou, Yuan Fang, and Xinming Zhang. Multigprompt for multi-task pre-training and prompting on graphs. *arXiv preprint arXiv:2312.03731*, 2023.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Werner Zellinger, Thomas Grubinger, Edwin Lughofer, Thomas Natschläger, and Susanne Saminger-Platz. Central moment discrepancy (cmd) for domain-invariant representation learning. In *International Conference on Learning Representations*, 2022.

Zhen Zhang, Meihan Liu, Anhui Wang, Hongyang Chen, Zhao Li, Jiajun Bu, and Bingsheng He. Collaborate to adapt: Source-free graph domain adaptation via bi-directional adaptation. In *Proceedings of the ACM on Web Conference 2024*, pp. 664–675, 2024.

Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Proceedings of the aaai conference on artificial intelligence*, volume 35, pp. 11015–11023, 2021.

Qi Zhu, Natalia Ponomareva, Jiawei Han, and Bryan Perozzi. Shift-robust gnns: Overcoming the limitations of localized graph training data. *Advances in Neural Information Processing Systems*, 34:27965–27977, 2021.

A Proof of Propositions

Theorem 1. *(Proved in GPF(Fang et al., 2024)) Given a pre-trained GNN model f, an input graph $\mathcal{G}$, for any graph-level transformation $g : \mathcal{G}\{\mathcal{A}, \mathcal{X}\} \rightarrow \mathcal{G}'\{\mathcal{A}', \mathcal{X}'\}$, there exists an additional extra feature vector ΔX that satisfies:*

$$f(\mathcal{A}, \mathcal{X} + \Delta X) = f(g(\mathcal{A}, \mathcal{X})) = f(\mathcal{A}', \mathcal{X}')$$

A.1 Proof of $\mathcal{L}_s$ in Main Paper Eq 4

Assuming that there is a test Graph: $\mathcal{G}_{te} = \{\mathcal{A}, \mathcal{X}\}$, and two augmented views of $\mathcal{G}_{te}$ generated from a stochastic function $Aug(\cdot)$, denoted by $\mathcal{G}_1$ and $\mathcal{G}_2$, a pre-trained GNN model f, then we have:

$$\mathcal{G}_1 = Aug(\mathcal{G}_{te}) = \{\mathcal{A}_1, \mathcal{X}_1\} \tag{Ap.1}$$
$$\mathcal{G}_2 = Aug(\mathcal{G}_{te}) = \{\mathcal{A}_2, \mathcal{X}_2\} \tag{Ap.2}$$

For arbitrary graph $\mathcal{G}_i = \{\mathcal{A}_i, \mathcal{X}_i\}$, let $\mathcal{G}_i^\star = \{\mathcal{A}_i^\star, \mathcal{X}_i^\star\}$ represents the input graph at which the loss function $\mathcal{L}(f(\mathcal{G}_i^\star), \mathcal{Y})$ is optimal. According to proposition A, $\mathcal{G}_{te}^\star = \mathcal{G}'^\star = \{\mathcal{A}_i, \mathcal{X}_i'^\star\}$. Therefore, there separately exists respective representation: $\Delta X_1^\star$, $\Delta X_2^\star$, and $\Delta X_{1,2}$, making the following formulas true:

$$f(\mathcal{A}_1, \mathcal{X}_1'^\star) = f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1^\star) \tag{Ap.3}$$
$$f(\mathcal{A}_2, \mathcal{X}_2'^\star) = f(\mathcal{A}_2, \mathcal{X}_2 + \Delta X_2^\star) \tag{Ap.4}$$
$$f(\mathcal{A}_2, \mathcal{X}_2) = f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_{1,2}) \tag{Ap.5}$$

Naturally, it is desirable to design a loss function so that an augmented view $\mathcal{G}'$ is close enough to the f-mapped representation in the representation space of the f mapped solution $\mathcal{G}'^\star$:

$$\begin{aligned} P_{A2S} &= \arg\min_{\Delta X_1} \mathbb{E}\left[\|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1) - f(\mathcal{A}_1, \mathcal{X}'^\star_1)\|^2\right] \\ &= \arg\min_{\Delta X_1} \mathbb{E}[\|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1)\|^2 - \\ &\quad 2f(\mathcal{A}_1, \mathcal{X}'^\star_1)^T f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1) + \\ &\quad \|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1^\star)\|^2]. \end{aligned} \tag{Ap.5}$$

The naive Augmentation-to-Augmentation target P_{A2A} with adaptation ΔX is defined as follows:

$$\begin{aligned} P_{A2A} &= \arg\min_{\Delta X_1} \mathbb{E}\left[\|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1) - f(\mathcal{A}_2, \mathcal{X}_2)\|^2\right] \\ &= \arg\min_{\Delta X_1} \mathbb{E}[\|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1)\|^2 - \\ &\quad 2f(\mathcal{A}_2, \mathcal{X}_2)^T f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1) + \\ &\quad \|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_{1,2})\|^2]. \end{aligned} \tag{Ap.6}$$

When

$$\mathbb{E}\left[f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_{1,2}) - f(\mathcal{A}_1, \mathcal{X}'^\star_1)\right] = 0 \ \wedge \ \mathbb{E}\left[\|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_1)\|^2 - \|f(\mathcal{A}_1, \mathcal{X}_1 + \Delta X_{1,2})\|^2\right] = 0, \tag{Ap.7}$$

we get $\mathbb{E}\left[f(g(\mathcal{G})) - f(\mathcal{G})\right] = 0$ in Eq 5.

B Algorithm

For detailed pseudocode under the augmented graphs number of $|\tau| = 2$, please refer to the right-hand side of the page.

Algorithm 1 GOAT for Full Test-Time Graph Adaptation

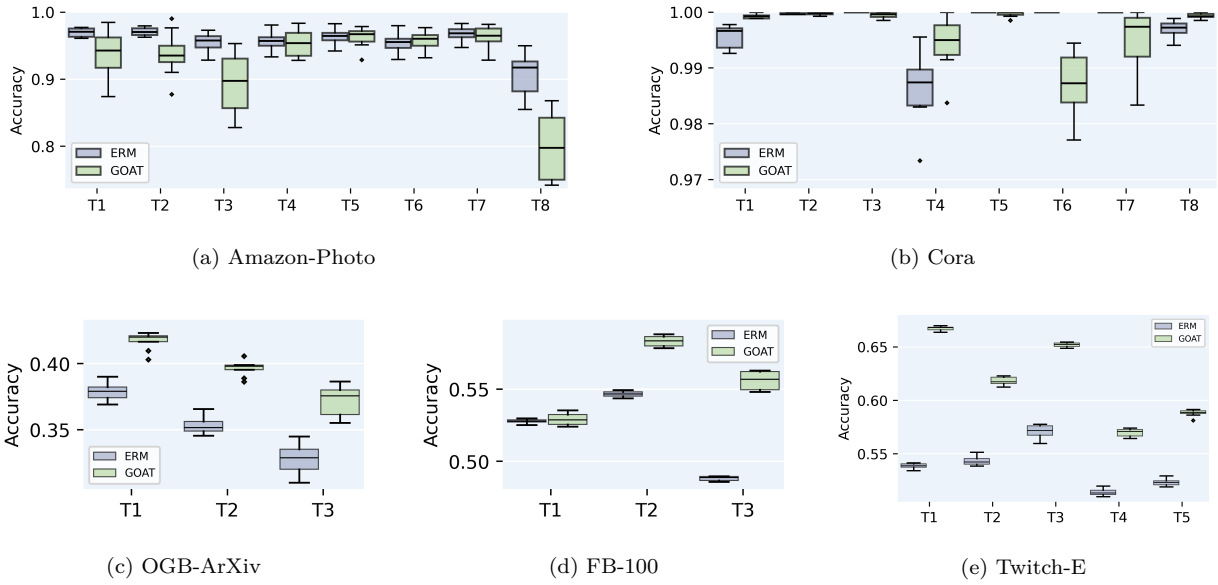
Input: Pre-trained GNN $f_{\theta^\star}$ ($\theta^\star$ is fixed, without the last layer which is the classifier head) and test graph $\mathcal{G}_{te} = \{\mathcal{A}, \mathcal{X}\}$, Sample method(DropEdge) $Aug(\cdot)$
Output: Model prediction $\hat{Y}$
Initialize $\text{LRA}_\psi, \alpha, \lambda$, learning rate η
$\mathcal{L}_{\text{best}} = \infty$, patience = k, $\text{patience}_{\text{now}} = 0$
for $t = 1$ to T **do**
  $\mathcal{G}' = DE(\mathcal{G}_{te}) = \{\mathcal{A}', \mathcal{X}'\}$, $\mathcal{G}'' = DE(\mathcal{G}_{te}) = \{\mathcal{A}'', \mathcal{X}''\}$
  $\Delta\hat{X}' = \text{LRA}_\psi(\mathcal{A}', \mathcal{X}')$, $\Delta\hat{X}'' = \text{LRA}_\psi(\mathcal{A}'', \mathcal{X}'')$
  $\Delta\hat{X}'_{emb} = f_{\theta^\star}(\mathcal{A}', \Delta\hat{X}')$, $\Delta\hat{X}''_{emb} = f_{\theta^\star}(\mathcal{A}'', \Delta\hat{X}'')$
  $z'_{p_{emb}} = f_{\theta^\star}(\mathcal{A}', \mathcal{X}'+\Delta\hat{X}')$, $z''_{p_{emb}} = f_{\theta^\star}(\mathcal{A}'', \mathcal{X}''+\Delta\hat{X}'')$
  $z'_{emb} = f_{\theta^\star}(\mathcal{A}', \mathcal{X}')$, $z''_{emb} = f_{\theta^\star}(\mathcal{A}'', \mathcal{X}'')$
  $\mathcal{L}_s = \mathbb{E}\|(z'_{p_{emb}} - z''_{emb}) + (z''_{p_{emb}} - z'_{emb})\|^2$
  $\mathcal{L}_c = \mathbb{E}\|(z'_{p_{emb}} - z'_{emb} - \Delta\hat{X}'_{emb}) + (z''_{p_{emb}} - z''_{emb} - \Delta\hat{X}''_{emb})\|^2$
  $\mathcal{L}_R = \mathbb{E}|\Delta\hat{X}'_{emb} + \Delta\hat{X}''_{emb}|$
  $\mathcal{L} = \alpha\lambda(\mathcal{L}_s + \mathcal{L}_c) + (1 - \lambda)\mathcal{L}_R$
  Update: $\psi \leftarrow \psi - \eta\Delta_\psi L$
  if $\mathcal{L}_R \leq \mathcal{L}_{best}$ **then**
    $\mathcal{L}_{best} = \mathcal{L}_R$
    $\text{patience}_{\text{now}} = 0$
  else
    $\text{patience}_{\text{now}} = \text{patience}_{\text{now}} + 1$
  end if
  if $\text{patience}_{\text{now}} \geq$patience **then**
    Stop
  end if
end for
$\Delta X = \text{LRA}_\psi(\mathcal{A}, \mathcal{X})$
$\hat{Y} = f_{\theta^\star}(\mathcal{A}, \mathcal{X} + \Delta X)$
return $\hat{Y}$

C Datasets and Hyper-Parameters

In this section, we reveal the details of reproducing the results in the experiments. We will release the source code upon acceptance.

C.1 Hyper-Parameter Setting

For the setup of backbone GNNs, we majorly followed EERM (Wu et al., 2023):



(a) Amazon-Photo (b) Cora

(c) OGB-ArXiv (d) FB-100 (e) Twitch-E

Figure 9: Classification performance on individual test graphs within each dataset for OOD setting.

(a) **GCN**: the architecture setup is 5 layers with 32 hidden units for Elliptic and OGB-ArXiv, and 2 layers with 32 hidden units for other datasets, with batch normalization for all datasets. The pre-train learning rate is set to 0.001 for Cora and Amz-Photo, 0.01 for other datasets; the weight decay is set to 0 for Elliptic and OGB-ArXiv, and 0.001 for other datasets.

(b) **GraphSAGE**: the architecture setup is 5 layers with 32 hidden units for Elliptic and OGB-ArXiv and 2 layers with 32 hidden units for other datasets, and with batch normalization for all datasets. The pre-train learning rate is set to 0.01 for all datasets; the weight decay is set to 0 for Elliptic and OGB-ArXiv, and 0.001 for other datasets.

(c) **GAT**: the architecture setup is 5 layers for Elliptic and OGB-ArXiv, and 2 layers for other datasets, with batch normalization for all datasets. Each layer contains 4 attention heads and each head is associated with 8 hidden units. The pre-train learning rate is set to 0.01 for all datasets; the weight decay is set to 0 for Elliptic and OGB-ArXiv, and 0.001 for other datasets.

(d) **GPR**: We use 10 propagation layers and 2 transformation layers with 32 hidden units. The pre-train learning rate is set to 0.01 for all datasets; the weight decay is set to 0 for Elliptic and OGB-ArXiv, and 0.001 for other datasets. Note that **GPR does not contain batch normalization layers**.

For the baseline methods, we tuned their hyper-parameters based on the validation performance. For Tent, we search the learning rate in the range of [1e-2, 1e-3, 1e-4, 1e-5] and the running epochs in [1, 10, 20, 30]. For EERM(Wu et al., 2023) and GTrans(Jin et al., 2023), we followed the instructions provided by the original paper. For GraphCTA(GCTA)(Zhang et al., 2024), we tune the feature adaptation η_1 in [5e-3, 1e-3, 1e-4, 1e-5, 1e-6], learning rate of structure adaptation η_2 in [0.5, 0.1, 0.01], and alternatively optimize node features epochs τ_1 in [1, 2, 3] and optimize graph structure epochs τ_2 in [1, 2], other parameters followed the instruction provided by the original paper. For GOAT , we adopt *DropEdge* as the augmentation function $\mathcal{DE}(\cdot)$ and set the drop ratio to 0.05, K-layer aggregation in LRA set to 1 due to some GNN only has two layers in some datasets while the last GNN layer performs as a classifier head. We use Adam Optimizer for LRA module tuning. We further search the learning rate η in [1e-2, 5e-3, 1e-3, 5e-4, 1e-4, 5e-5, 1e-5, 1e-6] for different backbones, the virtual nodes number $|n|$ in [1×, 2×, 5×, 10×, 20×] of the class number C, the attention dim d_{attn} in LRA in [2, 4, 8, 16, 32], total epochs T in [50, 100], and the patience in [1, 0.5, 0.1, 5e-2, 1e-2, 1e-3]. In the optimization target, we search the λ in [1, 3, 5, 10] and the α in [0.999, 0.9, 0.75,

Method	Amz-Photo	Cora	Elliptic	FB-100	OGB-ArXiv	Twitch-E
ERM	93.79±0.97	91.59±1.44	50.90±1.51	54.04±0.94	38.59±1.35	59.89±0.50
UDA-GCN	91.70±0.35	92.65±0.46	51.57±1.31	54.11±0.54	39.43±0.71	52.12±0.38
SRGNN	**94.64±0.17**	94.08±0.28	51.94±0.81	54.08±1.10	38.92±0.65	59.21±0.51
GOAT	94.35±1.32	**94.79±1.36**	**55.83±3.81**	**54.19±2.04**	**39.44±2.02**	**60.15±1.30**

Table 7: Performance comparison between GOAT with GCN and graph domain adaptation methods.

0.5, 0.25, 0.1, 5e-2, 1e-2, 5e-3]. We note that the process of tuning hyper-parameters is quick due to the high efficiency of test-time adaptation as we demonstrated in Section 4.1. Furthermore, not every test graph is learned over whole epochs set due to the patience of dissatisfaction of constraint in Eq 12.

Evaluation Protocol. For ERM (standard pre-training), we pre-train all the GNN backbones using the common cross-entropy loss. For EERM, it optimizes a bi-level problem to obtain a trained classifier. Note that the aforementioned two methods do not perform any test-time adaptation and their model parameters are fixed during the test. For the four test-time adaptation methods, Tent, GCTA, GTrans, and GOAT . We first obtain the GNN backbones pre-trained from ERM and adapt the model parameters or graph data at test time, respectively. Furthermore, Tent minimizes the entropy loss and GTrans and GCTA both minimize the contrastive surrogate loss, while GOAT minimizes the Target $\mathcal{L}_{A2A}$.

C.2 Hardware and Software Configurations

We perform experiments on NVIDIA GeForce RTX 3090 GPUs. The GPU memory and running time reported in Table 3 are measured on one single RTX 3090 GPU. Additionally, we use eight CPUs, with the model name Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz. The operating system utilized in our experiments was Ubuntu 22.04.3 LTS (codename jammy).

D More Experimental Results

D.1 Overall Comparison

To show the performance on individual test graphs, we choose SAGE as the backbone model and include the box plot on all test graphs within each dataset in Figure 9. We observe that GOAT generally improves over each test graph within each dataset, which validates the effectiveness of our proposed method.

D.2 Comparison to More Baseline and Backbones

To compare their empirical performance, we include two GraphDA methods (Zhu et al., 2021; Wu et al., 2020). SR-GNN regularizes the model's performance on the source and target domains. Note that SR-GNN was originally developed under the transductive setting where the training graph and test graph are the same. To apply SR-GNN in our OOD setting, we assume the test graph is available during the training stage of SR-GNN, as typically done in domain adaptation methods. UDA-GCN is another work that tackles graph data domain adaptation, which exploits local and global information for different domains. We followed the authors' suggestions in their paper to tune the hyper-parameters and the results are shown in Table 7. On the one hand, we can observe that these graph domain adaptation methods generally improve the performance of GCN under distribution shift and SRGNN is the best-performing baseline. On the other hand, GOAT performs the best on all datasets except Amz-Photo. On Amz-Photo, GOAT does not improve as much as SR-GNN, which indicates that joint optimization over source and target is necessary for this dataset. However, recall that domain adaptation methods are less efficient due to the joint optimization on source and target. Overall, the test-time graph adaptation with our adapter could better fit the specific distribution shifts that deviate from the source target. As shown in Table 8, GOAT could also adapt to more popular backbones.

Method	Amz-Photo	Cora	Elliptic	FB-100	OGB-ArXiv	Twitch-E
GTN	94.73±2.91	99.88±0.10	68.51±3.85	53.57±0.75	43.08±0.84	62.30±0.16
GTN + GOAT	94.75±2.97	99.85±0.12	70.08±2.50	54.94±0.61	44.11±0.84	63.79±0.27

Table 8: Performance comparison between GOAT with other backbones.

GraphID	$\mathcal{G}_0$	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_3$	$\mathcal{G}_4$	$\mathcal{G}_5$	$\mathcal{G}_6$	$\mathcal{G}_7$	$\mathcal{G}_8$
Amz-Photo	6.4	5.1	5.5	3.7	2.8	3.7	3.9	6.6	-
Cora	5.4	4.2	4.8	6.3	5.5	4.8	4.6	5.4	-
Elliptic	80.2	90.8	114.3	86.5	789.3	781.6	99.4	100.4	150.6
OGB-ArXiv	14.7	20.6	10.4	-	-	-	-	-	-
FB-100	29.7	16.9	32.9	-	-	-	-	-	-
Twitch-E	8.6	6.1	9.0	8.4	9.7	-	-	-	-

Table 9: CMD values on each individual graph based on the pre-trained GCN.

D.3 Quantifying distribution shift through LRA

In this section, we further show more distribution of ΔX generated by GOAT that indicates the OOD degree of each test graph. We can see an extreme shift in Figure 11(c) that as the snapshots flow from the validation, the mode and the mean value of ΔX shift away from the 0 initialized value, which shows a further deviation of the later test graphs from the train source distribution. Furthermore, following SR-GNN (Zhu et al., 2021), we adopt central moment discrepancy (CMD) (Zellinger et al., 2022) as the measurement to quantify the distribution shifts in different graphs, we present them in Table 9 as a comparison with ΔX in GOAT .

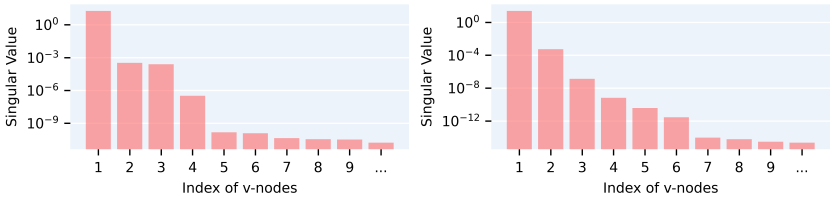
D.4 Low-rank of Node-level Representation on Large Graph

In Figure 10, we show other ΔX generated from LRA on two large test graphs in OGB-arXiv with 69499 and 120740 nodes.

E Ablation study

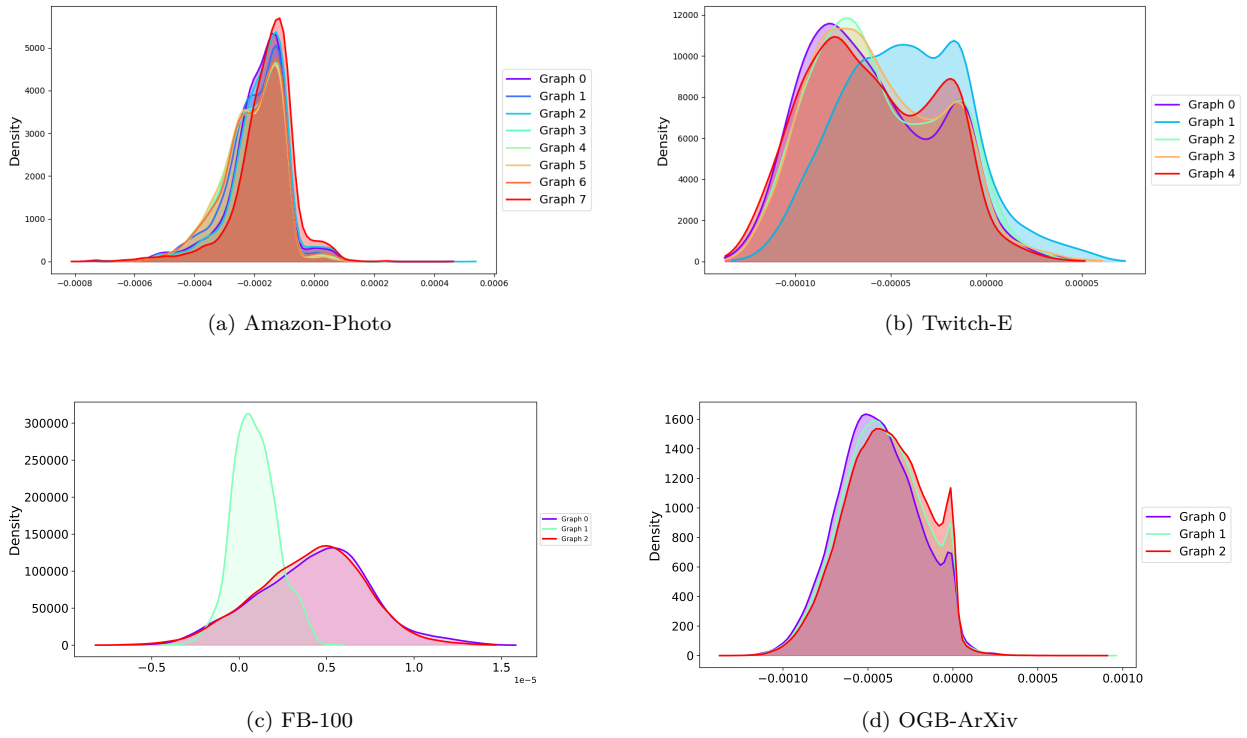
E.1 Optimization Object

In Figure 10, we show the parameter study of λ and α in $\mathcal{L}_{A2A}$. Noted that there could be a different proportion of $\frac{\lambda}{(1-\alpha)}$, while it still should have a rather value of α in that the constraint in Eq.(11) ($\mathcal{L}_R$) should be satisfied first then the other objective could work.



(a) $\mathcal{G}_1$ (b) $\mathcal{G}_2$

Figure 10: (a)(b) SVD of E in LRA after training on test graph $\mathcal{G}_1$, $\mathcal{G}_2$ in OGB-ArXiv with distribution shifts.



(a) Amazon-Photo

(b) Twitch-E

(c) FB-100

(d) OGB-ArXiv

Figure 11: ΔX Distribution after training on each dataset.

E.2 Different Augmentation Methods Used in Optimization

In Target 4, we used *DropEdge* as the augmentation function $Aug(\cdot)$ to obtain the augmented view and enough inductive graphs. In practice, the choice of augmentation can be flexible, and here we explore two other choices: node dropping and *FlipEdge* (You et al., 2020). Specifically, we adopt a ratio of 0.05 for node dropping, a ratio of 0.05 and 0.5 for *FlipEdge*, and ratios of 0.05 and 0.5 for *DropEdge*. We observe that (1) GOAT with any of the three augmentations can greatly improve the performance of GCN under distribution shift, and (2) different augmentations lead to slightly different performances on different datasets

E.3 Parameters in LRA

After tuning over all datasets, the hyperparameter almost shows a slight difference. Therefore, d_{attn} is set to 8, $|n|$ is almost $10 \times C$(C is the class number of nodes in test graph). Furthermore, we explore that, unlike the Transformers structure in NLP, the multi-head attention and the residual connection cannot improve the performance in our LRA module, which indicates the graph structure data information learned with GNNs has a different representation from that learned as in NLP as sequences.