Trustworthy anomaly detection for smart manufacturing

Li, Z.

Chapter 6

Cross-Domain Graph Level Anomaly Detection

Authors: **Zhong Li**, Sheng Liang, Jiayang Shi, Matthijs van Leeuwen

Published in IEEE Transactions on Knowledge and Data Engineering, 36(12), 7839–7850.

Abstract

Existing graph level anomaly detection methods are predominantly unsupervised due to high costs for obtaining labels, yielding sub-optimal detection accuracy when compared to supervised methods. Moreover, they heavily rely on the assumption that the training data exclusively consists of normal graphs. Hence, even the presence of a few anomalous graphs can lead to substantial performance degradation. To alleviate these problems, we propose a *cross-domain graph level anomaly detection method*, aiming to identify anomalous graphs from a set of unlabeled graphs (*target domain*) by using easily accessible normal graphs from a different but related domain (*source domain*). Our method consists of four components: a feature extractor that preserves semantic and topological information of individual graphs while incorporating the distance between different graphs; an adversarial domain classifier to make graph level representations domain-invariant; a one-class classifier to exploit label information in the source domain; and a class aligner to align classes from both domains based on pseudolabels. Experiments on seven benchmark datasets show that the proposed method largely outperforms state-of-the-art methods.

6.1 Introduction

Graph-structured data is ubiquitous, as it can represent relations between objects using edges and semantic characteristics of objects using node attributes. As a result, graph level anomaly detection has a wide range of potential applications, such as criminal detection in financial network [82], error detection in system logs [232], identifying specific molecules in drug discovery [233], and detecting unhealthy brain structures [234].

Graph neural networks (GNNs) are capable of learning discriminative feature representations for graphs and have significantly advanced benchmark results for graph level anomaly detection [79]. Similar to other types of neural networks, the impressive performance of graph neural networks (GNNs) is often attained by using a substantial amount of labeled data. However, the process of manually annotating graph data is laborious and thus often impractical. To circumvent this challenge, recent studies have turned to unsupervised (or semi-supervised) learning instead. These methods, however, strongly rely on the assumption that the training data exclusively consists of normal graphs. Our experiments demonstrate that even a minor presence of anomalous graphs in the training data can lead to substantial performance degradation for these methods (c.f. Table 6.3).

In practice, labeled data may be accessible or relatively cheap to obtain in some domains. Hence, in situations where a certain 'target' domain of interest suffers from a dearth of labeled data (or its purity cannot be guaranteed), there is a strong motivation to construct learners that can exploit abundant labeled data from a different but related domain.

Recent work on graph level anomaly detection [78, 79, 81, 82] is mostly unsupervised (or semi-supervised), and has been limited to detecting anomalies within a single domain. That is, the potential benefits of incorporating labeled information from a related domain has not yet been researched. In this paper we investigate how to transfer 'anomaly knowledge' from a source to a target graph database.

Unsupervised domain adaptation (UDA) is an attractive approach to achieve this: it adapts models learned from a source domain with plenty labeled data to a target domain without labels, and has demonstrated remarkable performance in computer vision and natural language processing [235, 236]. Although a few studies have explored UDA for cross-domain node classification, there has been no prior research on cross-domain graph level anomaly detection. Two challenges need to be overcome. First, most existing UDA methods are developed for vector-based data, such as im-

age and text data, for which a distance in a Euclidean space can be defined, while for graph-structured data distance is typically defined in a non-Euclidean space due to graph isomorphism. This makes directly applying off-the-shelf UDA methods to graphs impractical. Second, graph level anomaly detection is inherently more challenging than node level anomaly detection, as anomalies at the graph level may involve global patterns and interactions that cannot be easily discerned by examining individual nodes.

To fill this gap, being motivated and supported by domain adaptation theory [237], we propose an unsupervised domain adaptation based graph level anomaly detection method called ARMET. It addresses the following cross-domain graph level anomaly detection problem: given a target graph database with fully unlabeled graphs and a *different but related* source graph database that contains only normal graphs, learn a one-class classifier that identifies anomalous graphs from the target graph database.

To achieve this, ARMET leverages an adversarial learning approach consisting of four main components. First, to learn graph level representations, it utilizes a two-part feature extractor: a semantic feature extractor to jointly preserve the semantic and topological information of each graph, and a structure feature extractor to extract the structure of each graph domain. Second, a domain classifier is learned to make graph level representations domain-invariant, thereby reducing the domain discrepancy. Third, a one-class classifier is trained using normal source graphs, aiming to make the learned graph level representations label-discriminative. Finally, a class aligner is trained to align normal graphs in both domains while separating anomalous graphs and normal graphs in the target domain. As a result, in an end-to-end manner, ARMET can learn both domain-invariant and label-discriminative graph level representations, and thus effectively identify anomalous graphs from the target domain.

Our contributions can be summarized as follows:

- We introduce the cross-domain graph level anomaly detection problem, and develop ARMET, an effective approach to address this problem;
- ARMET is the first attempt to combine graph neural networks and adversarial domain adaptation techniques for performing graph level anomaly detection tasks;
- Experiments demonstrate its improved performance for graph level anomaly detection when compared to state-of-the-art unsupervised graph level anomaly detectors.

The rest of this work is organized as follows. Chapter 6.2 reviews related literature. Chapter 6.3 introduces relevant notations and formulates the research problem. Chapter 6.4 presents the framework of our method ARMET. Chapter 6.5 and 6.6 describe the design details of ARMET. Chapter 6.7 provides experimental setup, and Chapter 6.8 presents results and corresponding analysis. Chapter 6.10 concludes this work.

6.2 Related Work

We review work that is most closely related; readers are referred to the following surveys for more on transfer learning [238, 239], graph representation learning [188], and graph anomaly detection [240, 241].

Graph Level Anomaly Detection Unsupervised/semi-supervised methods include OCGIN [78], GLAM [79], GLocalKD [81], OCGTL [82] and CODEtect [84]. Unlike ARMET, CODEtect can only handle unattributed graphs. Meanwhile, OCGIN, GLAM, GLocalKD, and OCGTL can handle attributed graphs, but they heavily depend on the assumption that the training data exclusively contains normal graphs. This assumption is impractical or expensive in real-world applications. As we will show later, the presence of anomalies in the training data may largely decrease the performance of these methods (c.f. Table 6.3). Moreover, the supervised method iGAD requires fully labeled training data, which is expensive or sometimes impossible to obtain. In contrast, ARMET only requires that the source domain data exclusively contains normal graphs, while imposing no additional assumptions on the target domain data. Further, it is the first graph level anomaly detection method that leverages labeled data from a different but related domain.

Traditional Unsupervised Domain Adaptation Traditional UDA techniques, such as DeepCoral [242], DANN [243], ADDA [244], and CDAN [245], are primarily designed for addressing multi-class classification problems in computer vision and natural language processing. Consequently, these methods fail to account for the unique characteristics of graph-structured data, as well as the highly imbalanced nature of anomaly detection problems. Moreover, these methods assume that the source domain contains all classes and is fully labeled. As a result, their effectiveness diminishes when the source domain contains only partial classes (e.g., only the normal class in anomaly detection), and this will be shown later in Table 6.6.

Domain Adaptation on Graphs Most existing domain adaptation methods on graphs, such as SDA-DAGL [246], DANE [247], UDA-GCN [248], DASGA [249], ACDNE [250], CDNE [251], DANE [252], COMMANDER [253], AdaGCN [254], DGL

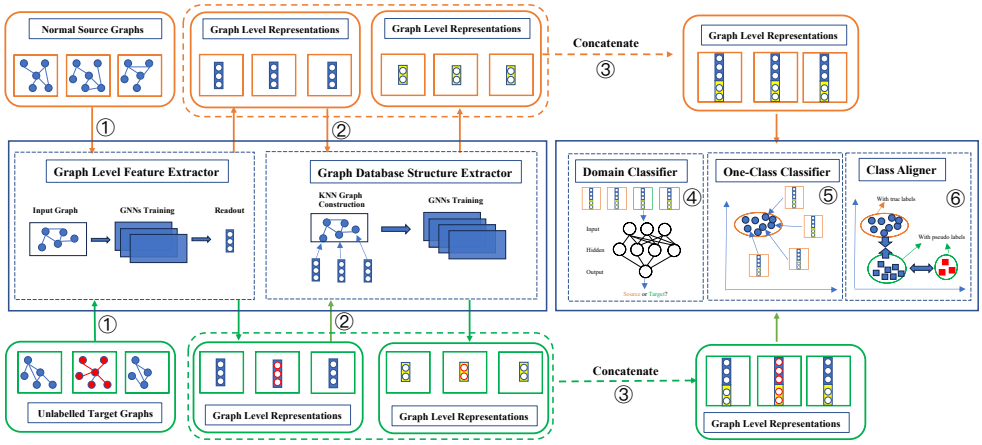

Figure 6.1: An overview of ARMET. Graphs and their learned representations are framed in oval rectangles, where the target domain is highlighted in green and the source domain in orange. Meanwhile, the semantic feature extractor, structure feature extractor, and other learners including one-class classifier, domain classifier, and class aligner are depicted in blue rectangles.

[255], AdaGIn [256] and CD-GAD [257], only consider domain adaptation from an individual graph to another graph.

Only a few works, including DGDA [258], CDA [259], and [260], consider domain adaptation from a set of graphs to another set of graphs. However, they focus on the multi-class graph classification problem, while ARMET is the first to consider cross-domain graph level anomaly detection, which is more challenging due to the extreme class imbalance. Moreover, unlike ARMET, these methods require the source domain data to contain all classes and be fully labeled.

6.3 Problem Statement

Following the notation commonly used in transfer learning [238], a *domain* $\mathcal{D}$ consists of two components, namely a feature space $\mathcal{X}$ and its marginal probability distribution $\mathbb{P}(\mathcal{X})$. Meanwhile, a *task* contains two components, that is, a label space $\mathcal{Y}$ and a predictive function $h(\cdot)$ that can be expressed as $\mathbb{P}(\mathcal{Y}|\mathcal{X})$. Moreover, we consider an attributed and undirected graph $G = (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V}$ is associated with a node feature matrix $\mathbf{X} \in \mathcal{R}^{|\mathcal{V}|\times C}$ and the the edge set $\mathcal{E}$ is associated with the adjacency matrix $\mathbf{A} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{V}|}$. In the following sections, we use the superscripts $(\cdot)^s$ and $(\cdot)^t$ to denote concepts in the source and target domains (also tasks), respectively.

Traditional Unsupervised Domain Adaptation on Graphs Traditional unsupervised domain adaptation (UDA) assumes that there is a set of labeled source graphs $\mathcal{G}^s = \{G_n^s, Y_n^s\}_{n=1}^{N_s}$ that are i.i.d drawn from $\mathbb{P}(\mathcal{X}^s, \mathcal{Y}^s)$, and another set of unlabeled target graphs $\mathcal{G}^t = \{G_n^s\}_{n=1}^{N_t}$ that are i.i.d drawn from $\mathbb{P}(\mathcal{X}^t)$. Moreover, UDA usually imposes the *covariate shift assumption*, i.e., $\mathbb{P}^s(\mathcal{X}^s) \neq \mathbb{P}^t(\mathcal{X}^t)$ but $\mathbb{P}^s(\mathcal{Y}^s|\mathcal{X}^s) = \mathbb{P}^t(\mathcal{Y}^t|\mathcal{X}^t)$. In other words, it assumes that the source and the target are different (in the sense that $\mathbb{P}^s(\mathcal{X}^s) \neq \mathbb{P}^t(\mathcal{X}^t)$) but related (in the sense that $\mathcal{X}^s = \mathcal{X}^t$ and $\mathbb{P}^s(\mathcal{Y}^s|\mathcal{X}^s) = \mathbb{P}^t(\mathcal{Y}^t|\mathcal{X}^t)$). On this basis, UDA aims to learn a classifier $h : \mathcal{X}^t \to \mathcal{Y}^t$ by using fully labeled source graphs and unlabeled target graphs.

For simplicity, we assume that the node feature matrices of the source (i.e., $\mathbf{X}^s$) and target graphs (i.e., $\mathbf{X}^t$) have the same dimensionality and their columns share the same semantic meanings. Otherwise, we can construct a unified node feature set $\mathcal{X} = \mathcal{X}^s \cup \mathcal{X}^t$ by following the practice in [254, 256]. This assumption is important for utilizing parameters-shared graph embedding models and fulfilling the covariate shift assumption in UDA.

Cross-Domain Graph Level Anomaly Detection The anomaly detection problem can be considered as a binary classification problem. Hence, we have $\mathcal{Y}^s = \mathcal{Y}^t = \{0, 1\}$, where 0 and 1 represent normal and abnormal graphs, respectively. To further relax the dependency on fully labeled source data, we assume that $\mathcal{G}^s$ only contains normal graphs. Specifically, given $\mathcal{G}^s = \{G_n^s, Y_n^s\}_{n=1}^{N_s} \in \mathcal{X}^s \times \mathcal{Y}^s$ with $Y_n^s = 0$ for $n \in 1, ..., N_s$, and $\mathcal{G}^t = \{G_n^t\}_{n=1}^{N_t} \in \mathcal{X}^t$, *Cross-Domain Graph Level Anomaly Detection* (CD-GLAD) aims to learn a binary classifier $g : \mathcal{X}^t \to \mathcal{Y}^t$ that accurately predicts anomaly labels for graphs in the target domain, with the assistance of both normal graphs from the source domain and unlabeled graphs from the target domain. Hence, the CD-GLAD problem is more practical but entails greater challenges than traditional UDA.

6.4 Proposed Method: ARMET

Motivation According to domain adaptation theory [237], given the source domain $\mathcal{D}^s = \{\mathcal{X}^s, \mathbb{P}(\mathcal{X}^s)\}$ and the target domain $\mathcal{D}^t = \{\mathcal{X}^t, \mathbb{P}(\mathcal{X}^t)\}$, given any binary classifier h drawn from a hypothesis class $\mathcal{H}$, for any $\delta \in (0, 1)$, with the probability at least of $1 - \delta$, we have

$$\epsilon^t(h) \leq \epsilon^s(h) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}^s, \mathcal{D}^t) + [\epsilon^s(h^*) + \epsilon^t(h^*)] + \omega, \tag{6.1}$$

where $\epsilon^t(h)$ and $\epsilon^s(h)$ indicate the expected error of classifier h on the target domain and source domain, respectively. Moreover, $d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}^s, \mathcal{D}^t)$ represents the domain discrepancy. Importantly, $[\epsilon^s h^*) + \epsilon^s(h^*)]$ means the combined error of the ideal joint classifier h^* on both domains, where $h^* =: \arg\min_{h\in\mathcal{H}}[\epsilon^s(h) + \epsilon^t(h)]$. Besides, ω is the item associated with the model complexity and the sample sizes.

In this work, we aim to learn a classifier h that has the minimal expected error on the target domain. Therefore, the sum of terms on the right side of (6.1) should be minimized. This sheds key insights into the design of our algorithm, which considers the following overall objective:

$$\mathcal{L}(\mathcal{X}^s, \mathcal{Y}^s, \mathcal{X}^t) = \lambda_1\mathcal{L}_{SC}(\mathcal{X}^s, \mathcal{Y}^s) - \lambda_2\mathcal{L}_{DA}(\mathcal{X}^s, \mathcal{X}^t) + \lambda_3\mathcal{L}_{CA}(\mathcal{X}^s, \mathcal{Y}^s, \mathcal{X}^t), \qquad (6.2)$$

where $\mathcal{L}(\mathcal{X}^s, \mathcal{Y}^s, \mathcal{X}^t)$ corresponds to the upper bound of $\epsilon^t(h)$, $\mathcal{L}_{SC}(\mathcal{X}^s, \mathcal{Y}^s)$ is the source classifier loss corresponding to $\epsilon^s(h)$, $\mathcal{L}_{DA}(\mathcal{X}^s, \mathcal{X}^t)$ is the domain classifier loss that approximates $d(\mathcal{D}^s, \mathcal{D}^t)$ (larger loss indicates smaller discrepancy), and $\mathcal{L}_{CA}(\mathcal{X}^s, \mathcal{Y}^s, \mathcal{X}^t)$ is the class alignment loss that corresponds to $[\epsilon^s(h^*) + \epsilon^s(h^*)]$. Further, the balance parameters $\lambda_1 > 0$, $\lambda_2 > 0$ and $\lambda_3 > 0$.

Approach Figure 6.1 depicts the architecture of our proposed method, dubbed ARMET (adversarial cross domain graph level anomaly detection). Specifically, ARMET adapts an adversarial learning framework to perform cross-domain graph level anomaly detection. It involves the four following main components:

- Parameters-Shared Feature Extractor h^{FE}: takes a source graph database $\mathcal{G}^s$ consisting of exclusively normal graphs and an unlabeled target graph database $\mathcal{G}^t$ as inputs, and aims to learn a representation vector $h^{FE}(G_i)$ for each graph $G_i \in \mathcal{G}^s \cup \mathcal{G}^t$ such that similar graphs (in terms of semantic and structure properties) from both domains have similar embeddings;

- One-Class Classifier h^s: takes the representation of each graph $h^{FE}(G_i)$ from the source domain as input, and learns a hypersphere to include embeddings of normal graphs while excluding those of anomalous graphs. This classifier can be directly used to predict labels for graphs in the target domain and corresponds to $\mathcal{L}_{SC}$;

- Domain Classifier h^d: takes the representation of each graph $h^{FE}(G_i)$ as input, and attempts to discriminate whether it is drawn from the source domain or the target domain such that the distributions of embeddings of both domains are aligned. And it corresponds to $\mathcal{L}_{DA}$;

- Class Aligner: takes $\{G_n^s, Y_n^s\}_{n=1}^{N_s}$ and $\{G_n^t, \hat{Y}_n^t\}_{n=1}^{N_t}$ as inputs, further making normal graphs from both domains have close embeddings, while normal graphs and anomalous graphs in the target domain have distant embeddings. Particularly, we apply the One-Class Classifier h^s to obtain pseudo-labels $\hat{Y}_n^t$ for graphs in the target domain while using the true labels Y_n^s for graphs in the source domain; This component corresponds to $\mathcal{L}_{CA}$.

For better organization, we divide these components into two modules, as shown in Figure 6.1: the *graph feature extraction module* that contains the feature extractor (steps ①,②, ③), and the *cross-domain anomaly detection module* that includes the domain classifier (step ④), the one-class classifier (step ⑤), and the class aligner (step ⑥). Importantly, the two modules are trained jointly in an end-to-end manner, although they are introduced separately in the following.

6.5 Module 1: Graph Feature Extraction

The graph feature extraction module consists of a feature extractor dubbed $h^{FE}(\cdot)$ with trainable parameters Θ^{FE}, aiming to extract graph level representations for graphs from source domain and target domain. This component is further decomposed to three sub-components: a semantic feature extractor (step ①) and a structure feature extractor (step ②), followed by a feature concatenation operator (step ③).

Semantic Feature Extractor We denote the semantic feature extractor as $h^{SE}(\cdot)$ with trainable parameters Θ^{SE}, which learns a graph level representation for each input graph. Specifically, we adapt a K-layer GIN model [73] followed by a READOUT function to obtain graph level representations due to the superior performance of GIN compared to other competing methods [261]. Concretely, GIN updates node representations as

$$f_v^{(k+1)} = \text{MLP}\left(\left(1+\alpha^{(k)}\right) f_v^{(k)} + \sum\nolimits_{u\in\mathcal{N}(v)} f_u^{(k)}\right), \tag{6.3}$$

where $f_v^{(k)}$ denotes the representation of node v learned at the k-th layer, $\mathcal{N}(v)$ indicates the neighbor set for node v, $\alpha^{(k)}$ is a trainable parameter while MLP represents a multilayer perceptron. Next, we obtain the graph level representation by leveraging $\text{READOUT}\left(f_v^{(k)}|k=1,...,K\right)$, which can be a simple permutation-invariant function such as the maximum, sum or mean. Using this, we can preserve the semantic and topological information of each graph.

Structure Feature Extractor We denote the structure extractor as $h^{ST}(\cdot)$ with parameters Θ^{ST}. We assume that $\mathcal{G}^s = \{G_1^s, ..., G_j^s, ..., G_N^s\}$ and $\mathcal{G}^t = \{G_1^t, ..., G_j^t, ..., G_M^t\}$ exhibit some inherent data structures such as clusters in $\mathcal{X}^s$ and $\mathcal{X}^t$, respectively. Inspired by [262], for each graph database, we construct a k-nearest neighbors graph (*KNN graph*) in the latent space to model its data structure, aiming to capture the neighborhood information between different graphs. Without loss of generality, we use $\mathcal{G}^s$ to demonstrate the construction of a *KNN graph*.

The *KNN graph* of $\mathcal{G}^s$ contains N nodes, with each node representing a source graph and its node attribute indicating the corresponding graph level representation vector learned by $h^{ST}(\cdot)$. Next, to generate edges, we can construct the adjacency matrix as

$$A_{ij} = \begin{cases} 1, & \text{if } G_i \in \mathcal{N}_k(G_j) \text{ or } G_j \in \mathcal{N}_k(G_i) \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

where $\mathcal{N}_k(G_j)$ represents the k-nearest neighbors set of graph G_j based on the Euclidean distance between graph level embeddings. After constructing the *KNN graph*, we can leverage another GIN model (without READOUT function) to learn the node representations, wherein a node represents a source graph instance.

Feature Concatenation Operator We utilize a concatenation operator that directly appends the structure feature for each graph to its semantic feature. Hence, this operator has no parameters. Overall, given a graph G_i, its final extracted feature can be expressed as $h^{FE}(G_i) = h^{SE}(G_i) \parallel h^{ST}(G_i)$. As a result, the corresponding trainable parameter $\Theta^{FE} = (\Theta^{SE}, \Theta^{ST})$. Importantly, these parameters are shared when learning representations for graphs from the source domain and the target domain, respectively.

6.6 Module 2: Cross-Domain Graph Level Anomaly Detection

This module contains three components: the one-class classifier, the domain classifier, and the class aligner. We first elucidate the rationale and design of each component, followed by introducing the theory of adversarial training.

One-Class Classifier: We use $h^s(\cdot)$ to represent the source classifier, which has no additional parameters beyond the feature extraction parameters Θ^{FE}. We train a one-class classifier on the source domain by minimizing the following One-Class Deep

SVDD objective [99]:

$$\mathcal{L}_{SC}(\mathcal{X}^s, \mathcal{Y}^s; \Theta^{FE}) =: \frac{1}{N_s} \sum_{m=1}^{N_s} \|h^{FE}(G_m) - \mathbf{o}\|_2^2, \tag{6.5}$$

where $h^{FE}(G_m)$ is the final extracted feature of graph G_m (from the source domain), and $\mathbf{o}$ is the learned center that represents the normality defined in the source domain. Minimizing this SVDD loss ensures the model learns the label information from the source domain.

Domain Classifier: We denote the domain classifier as $h^d(\cdot)$, with trainable parameters Θ^d in addition to parameters Θ^{FE}. We maximize the binary cross-entropy loss to learn domain-invariant features:

$$\mathcal{L}_{DA}(\mathcal{X}^s, \mathcal{X}^t; \Theta^d, \Theta^{FE}) =: \left[\frac{1}{N_s + N_t} \sum_{i=1}^{N_s+N_t} \left[d_i \log(\frac{1}{\hat{d}_i}) + (1 - d_i) \log(\frac{1}{1 - \hat{d}_i}) \right] \right], \tag{6.6}$$

where d_i denotes the binary ground-truth domain label for graph G_i. Specifically, d_i is 0 for graphs from the source domain and 1 for graphs from the target domain. Additionally, $\hat{d}_i$ represents the predicted probability that G_i belongs to the target domain, as determined by $h^d(\cdot)$. More precisely, the prediction $\hat{d}_i$ is given by $\hat{d}_i = h^d(h^{FE}(G_i))$, where $h^{FE}(\cdot)$ is the final feature extractor and $h^d(\cdot)$ is the domain classifier. In loss term (6.6), we consider the negative log-probability, expressed as $-\log(\hat{d}_i)$. This can be rewritten as $\log\left(\frac{1}{h^d(h^{FE}(G_i))}\right)$. The goal of this loss function is to maximize it, which encourages the feature extractor $h^{FE}(\cdot)$ to create similar representations for graphs from both the source and target domains. This helps align the domains, making the feature distributions of the source and target domains indistinguishable and reducing discrepancies between them.

Class Aligner: Structure consistency between domains (via parameters-shared feature extractor), discriminability in source domain (via source classifier), and domain-invariant features (via domain classifier) do not necessarily lead to discriminability in the target domain. That is, although we manage to align features cross domains, the learned features may be distorted in the sense that they are not representative of the underlying patterns in the target domain. As a result, features of the normal and abnormal classes in the target domain may exhibit close proximity or even overlap, leading to a high value of $\epsilon^t(h)$ in the target domain. This is known as excessive alignment in [263] and collapse of target neighborhood structure in [264].

To alleviate this problem, we should consider the third term in (6.1), namely $[\epsilon^s(h^*) + \epsilon^t(h^*)]$, that considers labels from both domains. Although the true label information in the target domain cannot be obtained, we can generate pseudolabels and minimize the class centroid alignment loss:

$$\mathcal{L}_{CA}(\mathcal{X}^s, \mathcal{Y}^s, \mathcal{X}^t, \hat{\mathcal{Y}}^t; \Theta^{FE}) =: \left[\psi(\mathcal{C}_n^s, \mathcal{C}_n^t) - \psi(\mathcal{C}_a^t, \mathcal{C}_n^t)\right], \tag{6.7}$$

where $\hat{\mathcal{Y}}^t$ are the **pseudolabels** obtained by directly applying $h^s(\cdot)$ to the target graphs, and $h^s(\cdot)$ is obtained by optimizing loss term (6.5) in previous iteration. Moreover, $\mathcal{C}_n^s$, $\mathcal{C}_n^t$, and $\mathcal{C}_a^t$ represent the centroid of normal source class, normal target class, and anomalous target class, respectively. The function $\psi(\cdot, \cdot)$ is a distance metric such as the Euclidean distance. Minimizing this loss can reduce the inter-domain distance between centroids of normal classes, while simultaneously maximizing the intra-domain distances between centroids of normal and abnormal classes within the target domain. Particularly, with an increase of training epochs, we expect that the discrepancy between source domain and target domain is reduced, the data structures are better aligned, the $h^s(\cdot)$ is further improved, and thus the pseudolabels are gradually updated to approach the ground-truth labels, progressively improving the class centroid alignment.

Adversarial Training: It can be seen that the optimization of (6.5) and (6.7) involves a minimization w.r.t. parameters Θ^{FE}, while the optimization of (6.6) concerns a maximization w.r.t. parameters Θ^d and Θ^{FE}. In other words, objective (6.6) competes against objectives (6.5) and (6.7) during training over the overall objective (6.2). To obtain a good trade-off, adversarial training techniques have been explored, with impressive results [243].

For simplicity, we rewrite $\mathcal{L}(\mathcal{X}^s, \mathcal{Y}^s, \mathcal{X}^t; \Theta^{FE}, \Theta^d)$ as $\mathcal{L}(\Theta^{FE}, \Theta^d)$, which contains two sets of parameters. Adversarial training attempts to find a saddle point $(\hat{\Theta}^{FE}, \hat{\Theta}^d)$ such that $\hat{\Theta}^{FE} = \underset{\Theta^{FE}}{\operatorname{argmin}}\, \mathcal{L}(\Theta^{FE}, \hat{\Theta}^d)$ and $\hat{\Theta}^d = \underset{\Theta^d}{\operatorname{argmax}}\, \mathcal{L}(\hat{\Theta}^{FE}, \Theta^d)$. In other words, we perform a minmax optimization over (6.2):

$$\min_{\Theta^{FE}} \max_{\Theta^d} \left[\lambda_1 \mathcal{L}_{SC}(\Theta^{FE}) - \lambda_2 \mathcal{L}_{DA}(\Theta^{FE}, \Theta^d) + \lambda_3 \mathcal{L}_{CA}(\Theta^{FE})\right]. \tag{6.8}$$

Hence, the trainable parameters can be optimized alternatively as follows:

$$\begin{aligned}\Theta^d &\leftarrow \Theta^d - \mu\frac{\partial \mathcal{L}_{DA}}{\partial \Theta^d},\\ \Theta^{FE} &\leftarrow \Theta^{FE} - \mu\left(\lambda_1\frac{\partial \mathcal{L}_{SC}}{\partial \Theta^{FE}} - \lambda_2\frac{\partial \mathcal{L}_{DA}}{\partial \Theta^{FE}} + \lambda_3\frac{\partial \mathcal{L}_{CA}}{\partial \Theta^{FE}}\right),\end{aligned} \tag{6.9}$$

where μ is the learning rate. We perform mini-batch training with gradient descent and the corresponding pseudo-code is provided in Algorithm 5.

Algorithm 5 Mini-batch Algorithm of ARMET

Input: Labeled source graphs $\mathcal{G}^s = \{G_n^s, Y_n^s\}_{n=1}^{N_s}$; Unlabeled target graphs $\mathcal{G}^t = \{G_n^s\}_{n=1}^{N_t}$; Balance parameters λ_1, λ_2 and λ_3; Batch size N_b; Maximal training epochs N_e; Maximal iteration per epoch N_i
Output: Predicted labels $\hat{\mathcal{Y}}^t$ of target graphs

Initialize parameters Θ^{FE}, Θ^d
for `epoch` $< N_e$ and not converge **do**
  for `iteration` $< N_i$ **do**
    Sample a batch $\mathcal{B}^s$ and $\mathcal{B}^t$
    Learn representations using h^{FE} for $\mathcal{B}^s$ and $\mathcal{B}^t$
    Compute source classifier loss $\mathcal{L}_{SC}$ using (6.5)
    Compute domain classifier loss $\mathcal{L}_{DC}$ using (6.6)
    Backpropagate $\mathcal{L}_{DC}$ and update Θ^d using (6.9)
    Compute class aligner loss $\mathcal{L}_{CA}$ using (6.7)
    Compute total loss $\mathcal{L}$ using (6.2)
    Backpropagate $\mathcal{L}$ and update Θ^{FE} using (6.9)
  end for
end for
$\hat{h}^{FE}(\mathcal{G}^t) \leftarrow$ Use feature extractor with optimized parameters $\hat{\Theta}^{FE}$ to extract features for $\mathcal{G}^t$
$\hat{\mathcal{Y}}^t \leftarrow$ Use optimized source classifier $\hat{h}^s(\cdot)$ to predict labels for $\hat{h}^{FE}(\mathcal{G}^t)$

6.7 Experiment Setup

We aim to answer the following research questions (RQ) via experiments:

RQ1 How does ARMET perform when compared to state-of-the-art graph level anomaly detection methods?

RQ2 How does each component of ARMET affect the performance? (Ablation study)

Table 6.1: Datasets. **#Nodes**, **#Edges**, **Degree**, **#Attr** denote the average number of nodes and edges, the average degree, and the dimensionality of node attributes, respectively.

Data	#Graphs	#Nodes	#Edges	Degree	#Attr
BG	5000(5%)	10	30	3	200
HD	5000(5%)	7	20	2.86	200
SP	5000(5%)	6	24	4	200
TB	5000(5%)	16	52	3.25	200
LL	500(10%)	4.7	4.5	0.96	2
LM	500(10%)	4.7	3.1	0.66	2
LH	500(10%)	4.7	3.3	0.70	2

RQ3 How does the performance of ARMET change with different hyperparameter values? (Sensitivity analysis)

In addition, to get a better understanding of ARMET, we utilize t-SNE [265] to visualize the learned graph representations from both domains.

6.7.1 Benchmark Datasets

As summarized in Table 6.1, we study the following datasets collected from various application fields of graph mining:

System Logs We use four benchmark datasets for log anomaly detection, as converting logs into graphs and then leveraging GNNs to detect anomalies can achieve superior performance [88, 232]. Hence, following [232], we construct four graph datasets: HDFS (HD) [64], BGL (BG), SPIRIT (SP), and THUNDERBIRD (TH) [123], where each dataset contains 5000 graphs with 5% anomalous graphs. Particularly, HD consists of Hadoop Distributed File System logs while BG, SP, and TH containing system logs collected from three different supercomputing systems. For this group of datasets, we create 12 transfer tasks.

Letter Drawings We use three benchmark graph datasets with letter drawings of varying levels of distortion: low (LL), medium (LM), and high (LH) [266]. Following the practice of downsampling classification datasets for anomaly detection [168], for each dataset the letters N, M, and W are selected as the normal class, comprising a total of 450 instances (150 instances per letter), while the letter F is chosen as the anomalous class (downsampled to 50 instances). For each graph, a node denotes the end point of a line, an edge represents a line, and a node attribute represents its two-dimensional coordinate. For these datasets, we create 6 transfer tasks.

Discussion on Dataset Selection: Some commonly used benchmark graph datasets, including BZR, DHFR and COX2 (small molecules), as well as IMDB-Binary and REDDIT (social networks), have been excluded from our analysis for the following reasons: 1) the UDA assumptions do not hold for them, as these datasets have different definitions of classification problem, namely they have different label semantics; 2) transferability cannot be guaranteed due to huge domain discrepancy between datasets. For instance, node attributes of graphs in these datasets have different dimensionalities and/or meanings, and these datasets have very different graph statistics; and 3) even supervised classification methods can only achieve very limited in-domain classification accuracy (i.e., with an accuracy lower than 0.7) on these datasets.

6.7.2 Baselines

We compare ARMET to the following baselines:

- **Unsupervised graph level anomaly detection methods:** We directly apply OCGIN [78], GLAM [79], and GLocalKD [81] on unlabeled target graphs.
- **Traditional Domain Adaptation Methods:** ADDA [244], CDAN [245], and DeepCoral [242] are state-of-the-art UDA methods for image classification. As they are not designed for graph-structured data, we modify these models for cross-domain graph level anomaly detection by following [257].

To explore the ceiling performance of GLAD methods, we additionally present the outcomes of iGAD [83], a supervised method that is not considered a direct competitor due to its reliance on labeled target data.

6.7.3 Evaluation

We employ the widely used Area Under the Curve of the Receiver Operating Characteristics curve (AUC ROC), Area Under the Curve of Precision Recall (AUC PR), and F1-Score to evaluate and compare the different methods, where a higher value (closer to 1) represents better anomaly detection accuracy. Particularly, we report the average values of AUC ROC, AUC PR, F1-Score and the corresponding standard deviations across 10 independent runs.

6.7.4 Implementation and Model Configuration

We use the publicly available implementations of OCGIN [1], GLAM [1], GLocalKD [2] and iGAD [3] with their recommended configurations. Besides, ADDA[4], CDAN [5] and DeepCoral [5] are adapted from their publicly available implementations. As they are not designed for graph-structured data, we modify these models for cross-domain graph level anomaly detection by following the practice in [257]:

- CDAN: We replace the AlexNet encoder with a GIN plus a mean readout function;
- ADDA: Similarly, we replace the CNN encoder with a GIN plus a mean readout function;
- DeepCoral: We replace the CNN encoder (CaffeNet) with a GIN plus a mean readout function and apply CORAL loss to the last classification layer.

For ARMET, by following the practice in [242], the values of hyperparameters λ_1, λ_2 and λ_3 can be set such that, after the training process (e.g., 100 epochs), the losses associated with one-class classification $\mathcal{L}_{SC}$, domain discrimination ($\mathcal{L}_{DA}$) and class-alignment ($\mathcal{L}_{CA}$) are approximately at the same magnitude (after being multiplied by their corresponding weights). The rational behind it is that we aim to learn feature representations that are both label-discriminative and domain-invariant [242]. Particularly, we found that there is usually not a single set of optimal weights for each transfer task, as pointed out in multi-task learning by [267]. Although this hyperparameter tuning method works well, it is time-consuming and labor-intensive. For simplicity, we can adapt a *de-facto* strategy for hyperparameter tuning in UDA, namely splitting the target dataset according to a ratio of 20% : 80%, where the 20% data with labels is used as the validation dataset to select these three hyperparameters and the remaining 80% data without labels is used as the test data.

For fair comparisons, all GIN models used in different domain adaptation methods are configured with the same backbone architecture, namely a backbone of two layers (64 hidden units) with each layer followed by a ReLU activation. Besides, all neural network based methods are trained using mini-batch gradient descent, with a batch-size of 512 on log anomaly detection datasets and a batchsize of 128 on other datasets

[1] https://github.com/lingxiaoshawn/glam
[2] https://github.com/RongrongMa/GLocalKD
[3] https://github.com/graph-level-anomalies/iGAD
[4] https://github.com/yuhui-zh15/pytorch-adda
[5] https://github.com/agrija9/deep-unsupervised-domain-adaptation

Table 6.2: Description of hyperparameters and their recommended values. **Range** indicates the values that we have tested in the sensitivity analysis, and boldfaced values represent the values suggested to use in the experiments.

Symbol	Meaning	Range
d	embedding dimension	{16, 32, **64**, 96, 128, 160, 192, 224, 256, 300}
k	number of neighbors in KDTree	{2, **5**, 8, 11, 14, 17, 20}
L	number of layers	$\{1, \mathbf{2}, 3, 4, 5\}$
Bs	batch size	{16, 32, 64, **128**, 256, **512**, 1024}
λ	weight decay parameter	$\{\mathbf{0.0001}, 0.001, 0.01, 0.1\}$
η	learning rate	$\{0.0001, 0.001, \mathbf{0.01}\}$
Re	readout function	**mean**, sum, max
Ep	Epochs for training	{**100**, 200, 300, 400}

respectively, initial learning rate of 0.01, weight decay rate of 0.0001, and a maximum of 200 training epochs. The settings for these hyperparameters for ARMET are summarized in Table 6.2. Other algorithm-specific hyperparamters are set in accordance with their respective references given by the original authors.

6.7.5 Training Hardware and Reproducibility

We implemented and ran all algorithms in Python 3.8, using PyTorch [109] and PyTorch Geometric [110] when applicable, on a workstation equipped with an Intel i7-11700KF CPU and Nvidia RTX3070 GPU. For reproducibility, all code and datasets are made available on GitHub[6].

6.8 Experiment Results and Analysis

We answer the three research questions as follows.

6.8.1 Detection Accuracy (RQ1)

We perform the following analysis based on the experiment results in Tables 6.3, 6.4 and 6.6. Particularly, the results in terms of AUC PR and F1-Score are consistent with those in terms of AUR ROC. Therefore, these results are either deferred to Table 6.8 or omitted.

[6]https://github.com/ZhongLIFR/ARMET/

Table 6.3: Anomaly detection accuracy (Average AUC ROC and corresponding standard deviations across 10 runs) of unsupervised methods (OCCIN, GLAM, and GLocalKD) under two scenarios: 1) when the training dataset is clean, namely containing exclusively normal graphs (shown on the left side of '→'), and 2) when the training dataset is contaminated, namely containing both normal and abnormal instances (shown on the right side of '→').

Dataset	OCGIN	GLAM	GLocalKD
BG	$0.93_{\pm 0.04} \rightarrow 0.71_{\pm 0.11}(\downarrow)$	$0.64_{\pm 0.09} \rightarrow 0.74_{\pm 0.06}(\uparrow)$	$0.93_{\pm 0.02} \rightarrow 0.91_{\pm 0.01}(\downarrow)$
SP	$0.62_{\pm 0.16} \rightarrow 0.59_{\pm 0.16}(\downarrow)$	$0.70_{\pm 0.09} \rightarrow 0.71_{\pm 0.09}(\uparrow)$	$0.81_{\pm 0.08} \rightarrow 0.66_{\pm 0.15}(\downarrow)$
HD	$0.98_{\pm 0.01} \rightarrow 0.88_{\pm 0.03}(\downarrow)$	$0.87_{\pm 0.06} \rightarrow 0.77_{\pm 0.09}(\downarrow)$	$0.45_{\pm 0.05} \rightarrow 0.45_{\pm 0.05}(-)$
TH	$0.81_{\pm 0.20} \rightarrow 0.43_{\pm 0.28}(\downarrow)$	$0.93_{\pm 0.05} \rightarrow 0.96_{\pm 0.09}(\uparrow)$	$0.93_{\pm 0.05} \rightarrow 0.76_{\pm 0.11}(\downarrow)$
LL	$0.99_{\pm 0.00} \rightarrow 0.70_{\pm 0.10}(\downarrow)$	$0.82_{\pm 0.18} \rightarrow 0.72_{\pm 0.18}(\downarrow)$	$0.56_{\pm 0.11} \rightarrow 0.53_{\pm 0.10}(\downarrow)$
LM	$0.91_{\pm 0.02} \rightarrow 0.64_{\pm 0.05}(\downarrow)$	$0.57_{\pm 0.10} \rightarrow 0.55_{\pm 0.07}(\downarrow)$	$0.56_{\pm 0.11} \rightarrow 0.53_{\pm 0.10}(\downarrow)$
LH	$0.65_{\pm 0.10} \rightarrow 0.52_{\pm 0.08}(\downarrow)$	$0.66_{\pm 0.08} \rightarrow 0.54_{\pm 0.10}(\downarrow)$	$0.56_{\pm 0.11} \rightarrow 0.53_{\pm 0.10}(\downarrow)$

Accuracy of Single-Domain GLAD

Table 6.3 demonstrates that unsupervised/semi-supervised methods OCGIN, GLAM[7], and GLocalKD generally suffer from large performance degradation when the training data is contaminated with anomalies. Moreover, these unsupervised/semi-supervised methods deliver very unstable results across different datasets even when the training data is clean. For example, GLocalKD yields high performance on BGL (ROC = 0.91), but very poor performance on HDFS (ROC = 0.45). In contrast, Table 6.4 shows that supervised method iGAD achieves perfect results, with an ROC of 1.0, on all cases. This indicates that these anomaly detection problems are solvable when sufficient labeled data is available in the target domain. However, this is unrealistic in many real-world scenarios as fully labeled data is usually expensive and often even impossible to obtain in practice.

Accuracy of Traditional UDA Methods

Table 6.4 shows that when there is only one class in the source domain, ADDA, CDAN, and DeepCoral behave like random guessing (e.g., with ROC $\approx$ 0.50), performing much worse than ARMET for most transfer tasks. The potential factors contributing to their subpar performance are as follows. First, CDAN considers the conditional distribution that captures the cross-variance between feature representations and classifier predictions, but the conditional distribution degenerates to the marginal distribution when there is only one class in the source domain. Second,

[7]With a few exceptions for GLAM, where the performance is slightly increased or stable.

Table 6.4: Anomaly detection accuracy (Average AUC ROC and corresponding standard deviations across 10 runs). The best and runner-up results are highlighted in **bold** and underlined, respectively. For unsupervised methods, we report the ROC values when the training dataset contains both normal and abnormal instances. For cross-domain methods, we report the ROC values when the source dataset contains only normal graphs. The results of supervised method are shown **only** to be able to put the performance of graph level anomaly detection methods in perspective (i.e., iGAD should **not** be considered as baseline). Moreover, the poor performance of traditional UDA methods (namely ADDA, CDAN and DeepCoral) are not due to intentionally choosing weak baselines or poor hyper-parameters. The underlying reasons for their subpar performance are given in Chapter 6.8.1.

Dataset	Supervised	Unsupervised			Traditional UDA			Ours
	iGAD	OCGIN	GLAM	GLocalKD	ADDA	CDAN	DeepCoral	ARMET
HD → BG					$0.50_{\pm 0.00}$	$0.50_{\pm 0.02}$	$0.50_{\pm 0.00}$	$\underline{0.77}_{\pm 0.06}$
SP → BG	$1.0_{\pm 0.00}$	$0.71_{\pm 0.11}$	$0.74_{\pm 0.06}$	$\mathbf{0.91}_{\pm 0.01}$	$0.50_{\pm 0.00}$	$0.49_{\pm 0.03}$	$0.50_{\pm 0.00}$	$\underline{0.85}_{\pm 0.07}$
TH → BG					$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\underline{0.83}_{\pm 0.05}$
BG → SP					$0.50_{\pm 0.00}$	$0.50_{\pm 0.01}$	$0.50_{\pm 0.00}$	$\mathbf{0.89}_{\pm 0.07}$
HD → SP	$1.0_{\pm 0.00}$	$0.59_{\pm 0.16}$	$0.71_{\pm 0.09}$	$0.66_{\pm 0.15}$	$0.50_{\pm 0.00}$	$0.51_{\pm 0.04}$	$0.50_{\pm 0.00}$	$\mathbf{0.91}_{\pm 0.04}$
TH → SP					$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\mathbf{0.88}_{\pm 0.06}$
BG → HD					$0.50_{\pm 0.00}$	$0.50_{\pm 0.01}$	$0.50_{\pm 0.00}$	$\underline{0.79}_{\pm 0.04}$
SP → HD	$1.0_{\pm 0.00}$	$0.88_{\pm 0.03}$	$0.77_{\pm 0.09}$	$0.45_{\pm 0.05}$	$0.50_{\pm 0.00}$	$0.51_{\pm 0.01}$	$0.50_{\pm 0.00}$	$\mathbf{0.90}_{\pm 0.03}$
TH → HD					$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\underline{0.81}_{\pm 0.06}$
BG → TH					$0.50_{\pm 0.00}$	$0.60_{\pm 0.20}$	$0.50_{\pm 0.00}$	$\mathbf{1.0}_{\pm 0.00}$
SP → TH	$1.0_{\pm 0.00}$	$0.43_{\pm 0.28}$	$0.96_{\pm 0.09}$	$0.76_{\pm 0.11}$	$0.50_{\pm 0.00}$	$0.69_{\pm 0.24}$	$0.50_{\pm 0.00}$	$\mathbf{1.0}_{\pm 0.00}$
HD → TH					$0.50_{\pm 0.00}$	$0.72_{\pm 0.24}$	$0.50_{\pm 0.00}$	$\mathbf{1.0}_{\pm 0.00}$
LM → LL	$1.0_{\pm 0.00}$	$0.70_{\pm 0.10}$	$0.72_{\pm 0.18}$	$0.53_{\pm 0.10}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\mathbf{0.98}_{\pm 0.02}$
LH → LL					$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\mathbf{0.96}_{\pm 0.00}$
LL → LM	$1.0_{\pm 0.00}$	$0.64_{\pm 0.05}$	$0.55_{\pm 0.07}$	$0.53_{\pm 0.10}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\mathbf{0.88}_{\pm 0.02}$
LH → LM					$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\mathbf{0.78}_{\pm 0.03}$
LL → LH	$1.0_{\pm 0.00}$	$0.52_{\pm 0.08}$	$0.54_{\pm 0.10}$	$0.53_{\pm 0.10}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\mathbf{0.79}_{\pm 0.02}$
LM → LH					$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$0.50_{\pm 0.00}$	$\mathbf{0.80}_{\pm 0.05}$

DeepCoral aligns the second-order statistics of layer activations in the source encoder and the target encoder, leading to random-guessing results since the source training data contains only normal graphs while the target training data includes both normal and abnormal graphs (and thus the statistics of layer activations should be different by nature). Third, ADDA performs poorly as it utilizes different encoders for the source and target domains, implying the importance of parameters-shared models when encoding graphs from two domains. The efficacy of the source encoder degrades to random-guessing when the source domain contains only a single class, as the source classifier fails to acquire any informative knowledge. Furthermore, these traditional UDA methods are primarily designed for balanced multiple classification problems, making them ill-suited for anomaly detection, which involves an extremely imbal-

Table 6.5: Ablation studies with the following stripped-down variations, where '✓' and '✗' mean the corresponding component is included or excluded, respectively.

Components	SD	/SF	/SC	/DC	/CA	ARMET
Structure Feature Extractor	✓	✗	✓	✓	✓	✓
One-Class Classifier	✓	✓	✗	✓	✓	✓
Domain Classifier	✗	✓	✓	✗	✓	✓
Class Aligner	✗	✓	✓	✓	✗	✓

anced binary classification task. As a reference, we report the performance of ADDA, CDAN, and DeepCoral when the source domain contains both labeled anomalous and normal instances in Table 6.6. One can see that the performance of ADDA, CDAN, and DeepCoral is largely boosted with auxiliary label information in most transfer tasks. However, even with auxiliary labels, they are still outperformed by ARMET in most cases.

Accuracy of ARMET

Table 6.4 shows that ARMET achieves the best performance on 13 out of 18 transfer tasks, and the second-best performance on the remaining tasks. Further, it largely outperforms unsupervised graph level anomaly detection methods on certain target datasets, demonstrating the benefit of leveraging label information from a different but related domain. For example, transferring knowledge from **HD** to **SP** leads to 54%, 28%, and 38% performance gains compared to OCGIN, GLAM, and GLocalKD, respectively. Finally, ARMET achieves impressive results under the setting that the source domain contains only normal graphs, while other traditional unsupervised domain adaptation methods provide "random-guessing" results, making ARMET more applicable to real-world scenarios.

6.8.2 Ablation Study (RQ2)

As summarized in Table 6.5, we here compare ARMET to five its stripped-down variations: SD is trained on the source domain and then directly applied on the target domain; /SF is ARMET trained without the structure feature extractor; /SC is ARMET trained without the source one-class classifier; /DC is ARMET trained without the domain classifier; and /CA is ARMET trained without the class aligner. We have the following important observations from Table 6.7:

- **SD vs ARMET.** ARMET is always superior to the case where we train the

Table 6.6: Anomaly detection accuracy (Average AUC ROC and corresponding standard deviations across 10 runs) of traditional UDA methods (including ADDA, CDAN, and DeepCoral) when the source domain contains both labeled anomalous and normal instances. ('↑' indicates the performance is boosted compared to the case where the source domain contains only normal instances, while '↓' means it is degraded.)

Dataset	ADDA*	CDAN*	DeepCoral*	ARMET
HD → BG	$0.50_{\pm 0.03}(-)$	$0.64_{\pm 0.11}(\uparrow)$	$0.51_{\pm 0.11}(\uparrow)$	$\mathbf{0.77}_{\pm 0.02}$
SP → BG	$0.47_{\pm 0.00}(\downarrow)$	$0.48_{\pm 0.06}(\downarrow)$	$0.49_{\pm 0.01}(\downarrow)$	$\mathbf{0.85}_{\pm 0.07}$
TH → BG	$0.49_{\pm 0.02}(\downarrow)$	$0.50_{\pm 0.00}(-)$	$0.50_{\pm 0.00}(-)$	$\mathbf{0.83}_{\pm 0.05}$
BG → SP	$0.56_{\pm 0.17}(\uparrow)$	$0.50_{\pm 0.08}(-)$	$0.49_{\pm 0.19}(\downarrow)$	$\mathbf{0.89}_{\pm 0.07}$
HD → SP	$0.63_{\pm 0.24}(\uparrow)$	$0.50_{\pm 0.03}(\downarrow)$	$0.60_{\pm 0.16}(\uparrow)$	$\mathbf{0.91}_{\pm 0.04}$
TH → SP	$0.54_{\pm 0.13}(\uparrow)$	$0.50_{\pm 0.00}(-)$	$0.50_{\pm 0.01}(-)$	$\mathbf{0.88}_{\pm 0.06}$
BG → HD	$0.62_{\pm 0.13}(\uparrow)$	$0.50_{\pm 0.08}(-)$	$0.48_{\pm 0.11}(\downarrow)$	$\mathbf{0.79}_{\pm 0.04}$
SP → HD	$0.57_{\pm 0.14}(\uparrow)$	$0.51_{\pm 0.03}(-)$	$0.51_{\pm 0.01}(\uparrow)$	$\mathbf{0.90}_{\pm 0.03}$
TH → HD	$0.57_{\pm 0.06}(\uparrow)$	$0.51_{\pm 0.02}(\uparrow)$	$0.50_{\pm 0.00}(-)$	$\mathbf{0.81}_{\pm 0.06}$
BG → TH	$0.48_{\pm 0.01}(\downarrow)$	$0.48_{\pm 0.20}(\downarrow)$	$0.54_{\pm 0.24}(\uparrow)$	$\mathbf{1.00}_{\pm 0.00}$
SP → TH	$0.53_{\pm 0.17}(\uparrow)$	$0.77_{\pm 0.24}(\uparrow)$	$0.73_{\pm 0.27}(\uparrow)$	$\mathbf{1.00}_{\pm 0.00}$
HD → TH	$0.58_{\pm 0.22}(\uparrow)$	$0.50_{\pm 0.17}(\downarrow)$	$0.84_{\pm 0.21}(\uparrow)$	$\mathbf{1.00}_{\pm 0.00}$
LM → LL	$0.93_{\pm 0.15}(\uparrow)$	$1.0_{\pm 0.00}(\uparrow)$	$1.0_{\pm 0.00}(\uparrow)$	$0.98_{\pm 0.02}$
LH → LL	$0.78_{\pm 0.23}(\uparrow)$	$0.99_{\pm 0.00}(\uparrow)$	$0.95_{\pm 0.05}(\uparrow)$	$0.96_{\pm 0.00}$
LL → LM	$0.53_{\pm 0.04}(\uparrow)$	$0.74_{\pm 0.00}(\uparrow)$	$0.63_{\pm 0.01}(\uparrow)$	$\mathbf{0.88}_{\pm 0.02}$
LH → LM	$0.65_{\pm 0.05}(\uparrow)$	$0.89_{\pm 0.00}(\uparrow)$	$0.76_{\pm 0.03}(\uparrow)$	$0.78_{\pm 0.03}$
LL → LH	$0.50_{\pm 0.01}(-)$	$0.78_{\pm 0.00}(\uparrow)$	$0.59_{\pm 0.02}(\uparrow)$	$\mathbf{0.79}_{\pm 0.02}$
LM → LH	$0.66_{\pm 0.08}(\uparrow)$	$0.87_{\pm 0.00}(\uparrow)$	$0.80_{\pm 0.01}(\uparrow)$	$0.80_{\pm 0.05}$

model on the source domain and then directly apply it to the target domain. This exemplifies the benefits and necessity of performing transfer learning.

- **/SF vs ARMET.** ARMET consistently outperforms its counterpart without the structural feature extractor. This underlines the importance of considering the neighborhood information in a set of graphs for CD-GLAD.
- **/SC vs ARMET.** It shows that explicitly incorporating the source label information via a source classifier is typically beneficial.
- **/DC vs ARMET.** In most cases, explicitly aligning the embeddings of both domains via a domain classifier is favorable. In certain cases, the presence of a domain classifier may have a small adverse impact on performance. One possible reason is that the domain classifier overly distorts the embedding space by aligning the embedding space in a brute-force manner. Another possible reason is that the embeddings are also implicitly aligned via the parameters-shared feature extractor and the class aligner.

Table 6.7: Ablation study (Average AUC ROC with 10 runs). SD: trained on source domain and then directly applied on target domain; /SF: ARMET without structure feature extractor; /SC: ARMET without source one-class classifier; /DC: ARMET without domain classifier; /CA: ARMET without class aligner.

Dataset	ARMET	SD	/SF	/SC	/DC	/CA
HD → BG	0.77	0.55	0.74	0.48	0.55	0.57
SP → BG	0.85	0.66	0.81	0.55	0.60	0.58
TH → BG	0.83	0.60	0.74	0.64	0.64	0.77
BG → SP	0.89	0.71	0.55	0.67	0.62	0.59
HD → SP	0.91	0.74	0.62	0.73	0.72	0.64
TH → SP	0.88	0.61	0.52	0.73	0.77	0.58
BG → HD	0.79	0.60	0.58	0.59	0.59	0.60
SP → HD	0.90	0.61	0.74	0.67	0.53	0.67
TH → HD	0.81	0.68	0.66	0.70	0.64	0.68
BG → TH	1.0	0.77	0.61	1.0	0.90	0.73
SP → TH	1.0	0.56	0.06	1.0	0.77	0.68
HD → TH	1.0	0.88	0.79	1.0	0.96	0.88
LM → LL	0.98	0.98	0.71	0.70	0.99	0.98
LH → LL	0.96	0.92	0.65	0.73	0.87	0.90
LL → LM	0.88	0.74	0.75	0.62	0.77	0.85
LH → LM	0.78	0.76	0.55	0.42	0.71	0.73
LL → LH	0.79	0.74	0.60	0.59	0.72	0.71
LM → LH	0.80	0.71	0.56	0.63	0.67	0.70

- **/CA vs ARMET.** The removal of the class aligner always leads to a performance degradation. This corroborates the importance of considering labels (or pseudo-labels) in the target domain when performing CD-GLAD.

6.8.3 Sensitivity Analysis (RQ3)

We examine the effects of the following hyperparameters on the detection performance, and Figure 6.2 depicts the selected results on four representative transfer tasks:

- The number of embedding dimensions d in parameters-shared feature extractor: most transfer tasks can achieve the best performance with an embedding dimension of 64. Small fluctuations can be observed with varying number of dimensions. Moreover, an overly small value of d may lead to suboptimal performance, while a large value introduces a considerable computational burden.

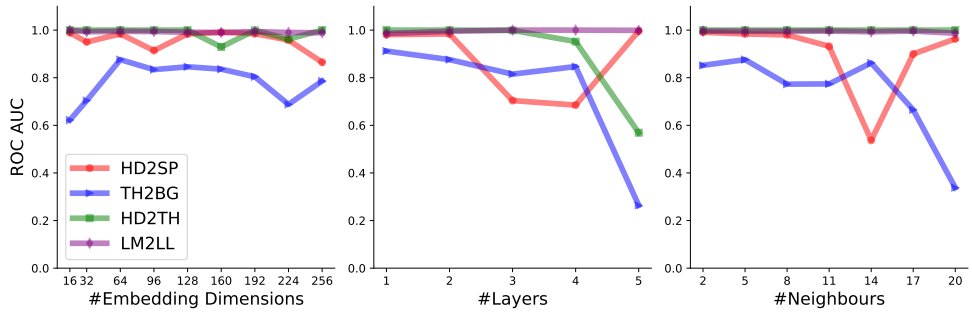

Figure 6.2: Sensitivity analysis of hyperparameters on four representative transfer tasks (HD → SP, TH → BG, HD → TH, LM → LL). Average results over five runs.

- The number of hidden layers L in the GIN model in the parameters-shared feature extractor: optimal performances are obtained when $L = 1$ or $L = 2$ on most transfer tasks. A further increase of its value usually results in performance degradation, which is widely known as over-smoothing [268].
- The number of neighbors k in the KNN graph: most transfer tasks can obtain the best performance on a wide range of k's values (namely $1 \leq k \leq 10$). However, further increasing the value of k may cause large performance fluctuations.

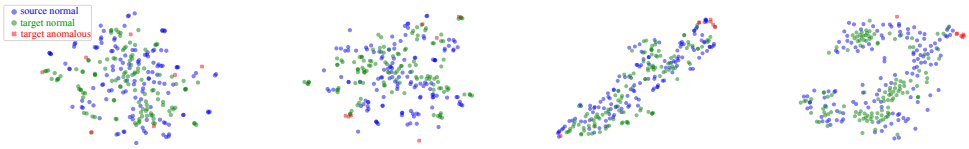

Figure 6.3: T-SNE visualization of the graph representation space during domain adaptation on task **LM** → **LL**. From left to right: the number of epochs correspond to zero, two, five and one hundred, respectively.

6.8.4 Visualization with T-SNE

Figure 6.3 visualizes the domain alignment and anomaly separation process on the transfer task **LM** → **LL**. The target normal graphs (green dots) are progressively adapted to the source normal graphs (blue dots), while the target anomalous graphs (red crosses) become increasingly separable in the embedding space.

6.9 Discussion on Transferability

As shown in Table 6.7, when comparing ARMET with its counterpart without explicit transfer learning (namely the column 'SD' that represents the scenarios where we train the model on the source domain and then directly apply it to the target domain), the performance gains from transfer learning are consistently non-negative. Moreover, when comparing ARMET with the best performing single-domain graph level anomaly detectors that are trained directly on the target domain, the performance gains are often positive (in 13 out 18 cases, see Table 6.4). This further demonstrates the benefits of transfer learning.

The underlying reason why transfer learning is beneficial in this context is that the extent of "relatedness" among System Logs datasets (i.e., BG, HD, SP and TH) is large enough, and so is the extent of "relatedness" among Letter Drawings datasets (i.e., LL, LM, and LH). In cross domain graph-level anomaly detection, it is crucial to ensure that the semantic meanings of normal patterns in the source and target domains are approximately the same. For instance, in System Logs data, these are normal patterns of system operations, while in Letter Drawings data, they represent the same set of letters.

As in other typical unsupervised domain adaptation settings, we assume that the source and target domains are different yet related. However, the extent of "relatedness" in this study is ensured based on our domain knowledge rather than a quantifiable transferability metric. To our knowledge, how to quantify the extent of this "relatedness" (namely *transferability* cross datasets) remains a long-standing problem [269]. Notably, when this "relatedness" is low, it may introduce negative transfer [270]. Moreover, it is critical to note that the transferability is usually *asymmetrical*, e.g., the performance gain for transfer task SP $\rightarrow$ TH is 0.44 (namely 1.0 minus 0.56 in terms of ROC AUC), which is different from the performance gain for TH $\rightarrow$ SP (namely 0.23 in terms of ROC AUC). Therefore, we should exercise caution (especially in safety-critical fields such as healthcare) when the transferability from the source domain to the target domain cannot be guaranteed, whether from a quantifiable metric or domain knowledge perspective.

6.10 Conclusions

This paper studies the problem of cross-domain graph level anomaly detection, wherein a set of unlabeled graphs from the target domain and a set of normal graphs from a

different but related domain are given. We propose ARMET, a theoretically motivated, novel method to solve this widely encountered but largely understudied problem. Specifically, ARMET consists of four components: a feature extractor, an adversarial domain classifier, a one-class classifier, and a class aligner. It is the first attempt to combine graph neural networks and adversarial domain adaptation techniques for performing graph level anomaly detection tasks. Extensive experiments demonstrate the efficacy of ARMET and its superiority to single-domain graph level anomaly detection methods and traditional unsupervised domain adaptation methods. Moreover, extensive ablation studies validate the benefits of incorporating each component into ARMET. Understanding and quantifying the transferability across different graph domains should be addressed in future research.

Table 6.8: Anomaly detection accuracy: average **AUC PR** and corresponding standard deviations across 10 runs (Top), and average **F1-Score**(Binary) and corresponding standard deviations across 10 runs (Bottom)

Dataset	Supervised	Unsupervised			Traditional UDA			Ours
	iGAD	OCGIN	GLAM	GLocalKD	ADDA	CDAN	DeepCoral	ARMET
HD → BG					$0.05_{\pm 0.00}$	$0.05_{\pm 0.01}$	$0.05_{\pm 0.00}$	$0.14_{\pm 0.09}$
SP → BG	$1.0_{\pm 0.00}$	$0.19_{\pm 0.09}$	$0.14_{\pm 0.03}$	$\mathbf{0.46}_{\pm 0.10}$	$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$\underline{0.25}_{\pm 0.13}$
TH → BG					$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$\underline{0.29}_{\pm 0.11}$
BG → SP					$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$\mathbf{0.76}_{\pm 0.12}$
HD → SP	$1.0_{\pm 0.00}$	$0.08_{\pm 0.03}$	$0.09_{\pm 0.02}$	$0.06_{\pm 0.01}$	$0.05_{\pm 0.00}$	$0.09_{\pm 0.08}$	$0.05_{\pm 0.00}$	$\mathbf{0.83}_{\pm 0.15}$
TH → SP					$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$\mathbf{0.90}_{\pm 0.05}$
BG → HD					$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$\underline{0.48}_{\pm 0.15}$
SP → HD	$1.0_{\pm 0.00}$	$0.43_{\pm 0.06}$	$\mathbf{0.62}_{\pm 0.07}$	$0.40_{\pm 0.04}$	$0.05_{\pm 0.00}$	$0.06_{\pm 0.02}$	$0.05_{\pm 0.00}$	$0.29_{\pm 0.10}$
TH → HD					$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$0.05_{\pm 0.00}$	$\underline{0.49}_{\pm 0.10}$
BG → TH					$0.06_{\pm 0.00}$	$0.06_{\pm 0.00}$	$0.06_{\pm 0.00}$	$\mathbf{0.93}_{\pm 0.13}$
SP → TH	$1.0_{\pm 0.00}$	$0.12_{\pm 0.15}$	$0.51_{\pm 0.14}$	$0.08_{\pm 0.02}$	$0.06_{\pm 0.00}$	$0.19_{\pm 0.29}$	$0.06_{\pm 0.00}$	$\mathbf{1.0}_{\pm 0.00}$
HD → TH					$0.06_{\pm 0.00}$	$0.41_{\pm 0.48}$	$0.06_{\pm 0.00}$	$\mathbf{0.98}_{\pm 0.01}$
LM → LL	$1.0_{\pm 0.00}$	$0.19_{\pm 0.06}$	$0.42_{\pm 0.20}$	$0.13_{\pm 0.03}$	$0.03_{\pm 0.00}$	$0.03_{\pm 0.00}$	$0.03_{\pm 0.00}$	$\mathbf{0.92}_{\pm 0.06}$
LH → LL					$0.03_{\pm 0.00}$	$0.03_{\pm 0.00}$	$0.03_{\pm 0.00}$	$\mathbf{0.51}_{\pm 0.15}$
LL → LM	$1.0_{\pm 0.00}$	$0.36_{\pm 0.04}$	$0.30_{\pm 0.07}$	$0.13_{\pm 0.03}$	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$\mathbf{0.62}_{\pm 0.11}$
LH → LM					$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$\mathbf{0.37}_{\pm 0.05}$
LL → LH	$1.0_{\pm 0.00}$	$0.25_{\pm 0.04}$	$0.39_{\pm 0.09}$	$0.13_{\pm 0.03}$	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$\mathbf{0.43}_{\pm 0.08}$
LM → LH					$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$0.10_{\pm 0.00}$	$\mathbf{0.47}_{\pm 0.12}$

HD → BG					$0.00_{\pm 0.00}$	$0.03_{\pm 0.02}$	$0.00_{\pm 0.00}$	$0.08_{\pm 0.15}$
SP → BG	$1.0_{\pm 0.00}$	$0.15_{\pm 0.08}$	$0.18_{\pm 0.09}$	$\mathbf{0.34}_{\pm 0.06}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.15_{\pm 0.18}$
TH → BG					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\underline{0.29}_{\pm 0.11}$
BG → SP					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.76}_{\pm 0.18}$
HD → SP	$1.0_{\pm 0.00}$	$0.05_{\pm 0.08}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.09_{\pm 0.18}$	$0.00_{\pm 0.00}$	$\mathbf{0.79}_{\pm 0.39}$
TH → SP					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.85}_{\pm 0.06}$
BG → HD					$0.00_{\pm 0.00}$	$0.02_{\pm 0.04}$	$0.00_{\pm 0.00}$	$\underline{0.50}_{\pm 0.13}$
SP → HD	$1.0_{\pm 0.00}$	$0.47_{\pm 0.07}$	$\mathbf{0.56}_{\pm 0.10}$	$0.26_{\pm 0.02}$	$0.00_{\pm 0.00}$	$0.04_{\pm 0.06}$	$0.00_{\pm 0.00}$	$0.09_{\pm 0.09}$
TH → HD					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\underline{0.50}_{\pm 0.10}$
BG → TH					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.97}_{\pm 0.05}$
SP → TH	$1.0_{\pm 0.00}$	$0.07_{\pm 0.22}$	$0.55_{\pm 0.32}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.17_{\pm 0.37}$	$0.00_{\pm 0.00}$	$\mathbf{1.0}_{\pm 0.00}$
HD → TH					$0.00_{\pm 0.00}$	$0.39_{\pm 0.53}$	$0.00_{\pm 0.00}$	$\mathbf{0.99}_{\pm 0.00}$
LM → LL	$0.99_{\pm 0.01}$	$0.14_{\pm 0.07}$	$0.39_{\pm 0.21}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.91}_{\pm 0.01}$
LH → LL					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.57}_{\pm 0.10}$
LL → LM	$0.99_{\pm 0.01}$	$0.40_{\pm 0.03}$	$0.29_{\pm 0.09}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.62}_{\pm 0.11}$
LH → LM					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.41}_{\pm 0.07}$
LL → LH	$0.99_{\pm 0.01}$	$0.25_{\pm 0.08}$	$0.37_{\pm 0.09}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.49}_{\pm 0.06}$
LM → LH					$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$\mathbf{0.48}_{\pm 0.12}$