Inductive Anomaly Detection in Dynamic Graphs with Accumulative Causal Walk Alignment

Leshanshui Yang[1,2] (✉), Clément Chatelain[3], and Sébastien Adam[2]

[1] Saagie, 72 Rue de la République, 76140 Le Petit-Quevilly, France
[2] Univ Rouen Normandie, LITIS UR 4108, F-76000 Rouen, France
[3] INSA Rouen Normandie, LITIS UR 4108, F-76000 Rouen, France

Abstract. Leveraging structural information and temporal evolution, dynamic graph-based methods have advanced anomaly detection, addressing challenges in real-world social and transaction networks. Despite progress, existing approaches, particularly those based on Graph Neural Networks, face challenges with long-range dependencies and encoding unseen nodes, affecting model performance and generalisation. This paper presents the Dual-Contextual Inductive Dynamic Graph Transformer (DCIDGT), an architecture that efficiently encodes both global and local contexts of target edges. Through a novel mechanism, Accumulative Causal Walk Alignment, DCIDGT captures global spatio-temporal information for unseen nodes and ensures their semantic alignment across snapshots. Our approach is evaluated on real-world cryptocurrency transaction datasets with PR AUC metric that significantly outperforms existing baselines, demonstrating its effectiveness and potential in dynamic graph representation learning.

Keywords: Dynamic graph representation learning · Anomaly detection · Inductive learning · Transformer.

1 Introduction

Anomaly detection tasks focus on identifying anomalous patterns that deviate from typical observations [13]. Over the last decade, graph-based methods have contributed significantly to the development of anomaly detection, by exploiting the expressive power of structural information. More recently, these graph-based approaches have been extended to dynamic graph (DG) scenarios where the graph structure and the graph attributes evolve [13,20] over time. These advances have pushed the boundaries of anomaly detection and efficiently solved real-world problems [4,19,42]. As example, with the explosion of cryptocurrencies and social media platforms, the detection of anomalous edges in DGs has applications in a variety of areas, including email systems [42], transaction networks [4] and, more broadly, social networks [40].

To address the Dynamic Graph Anomaly Detection problem, early approaches relied on handcrafted features to predict anomalous edges or nodes, which is a time-consuming and labour-intensive process [13,20]. More recently, deep learning models, including in particular Graph Neural Networks (GNN) [42,4], have been employed to efficiently learn graph representations and make predictions. However, Dynamic Graph Anomaly Detection faces two major challenges: **long-range dependencies** and **unseen node encoding**.

Over-squashing is a common problem faced by GNNs: When the propagation path of information is long or a node has many neighbours, compressing all this information into a fixed-size representation leads to loss and distortion, severely affecting the model's performance and generalisation ability [1,33].

Another major challenge is encoding newly appearing nodes, a long-standing issue in the field of dynamic graph learning known as "Inductive Learning" and referred to as the "Cold-Start" problem in industry [3,37]. This problem is particularly challenging in networks without node attributes, such as cryptocurrency trading network [18,17], which demand extracting representations directly from the graph structure when nodes first appear. To our knowledge, the most recent approaches avoid using graph-level representation of unseen nodes and instead, attempt to encode local information about nodes by computing statistical properties of subgraphs [4,19]. However, these approaches fall short in capturing of the long-range dependencies in both spatial and temporal aspects. Consequently, encoding dynamic graphs in the inductive setting with simultaneous local and graph-level focus in both temporal and spatial contexts is a challenging problem.

To address these two challenges, this paper presents two contributions to the field of dynamic graph edge anomaly detection. As opposed to the state-of-the-art algorithms mentioned above which focus on subgraphs, we introduce Dual-Contextual Inductive Dynamic Graph Transformer (**DCIDGT**). The model architecture is designed to efficiently encode both long-range and local spatio-temporal information, providing a rich foundation of global and historical insights for downstream tasks. To encode unseen nodes and capture global spatio-temporal information, we propose a novel mechanism called Accumulative Causal Walk Alignment (**ACWA**). This mechanism incorporates historical information in causal random walks to obtain global node embeddings at each snapshot, and then aligns their semantics across snapshots using the Orthogonal Procruste method [12] optimised for DGs. Through experiments on the edge anomaly detection task using publicly available real-world datasets [18,17], our model outperforms existing baseline models. The paper is structured as follows. Section 2 gives important definitions and describes related works. Section 3 presents the main contributions of the paper while section 4 describes the experimental part of the work and discusses obtained results.

2 Preliminaries and related works

In the nascent field of Dynamic Graph Anomaly Detection, many of the edge anomaly detection approaches use discrete-time dynamic graphs (DTDG) for representing DGs [40,42,4,19]. In this case, the dynamic graph is represented by a series of snapshots (Definition 1), and the system learns to predict the edge anomaly score based on historical information (Definition 2). Figure 1 describes the edge anomaly detection task on DTDG.

Definition 1 (Discrete Time Dynamic Graphs). *Discrete Time Dynamic Graphs (DTDGs) can be described by a sequence of graphs* $\mathbb{G} = \{G^1, G^2, \ldots, G^T\}$. *Each snapshot* G^t *at time step* t *comprises a set of edges* $E^t \subseteq \mathbb{E}$ *where* $\mathbb{E}$ *denotes the whole set of edges over the* T *time steps. The set of nodes involved in* E^t *is denoted as* $V^t \subseteq \mathbb{V}$, $\mathbb{V}$ *being the whole set of nodes over the* T *time steps.* $\mathbb{G}$ *is fully described by* $\{E^1, E^2, \ldots, E^T\}$. $N^t = |V^t|$ *and* $M^t = |E^t|$ *are respectively the number of nodes and of edges snapshot* G^t.

Definition 2 (Edge Anomaly Detection in DTDGs). *Edge Anomaly Detection in DTDGs aims to predict whether an edge is an anomaly for each edge in* E^t *at time step* t, *based on the given previous and current information* $\{E^1, \ldots, E^t\}$.

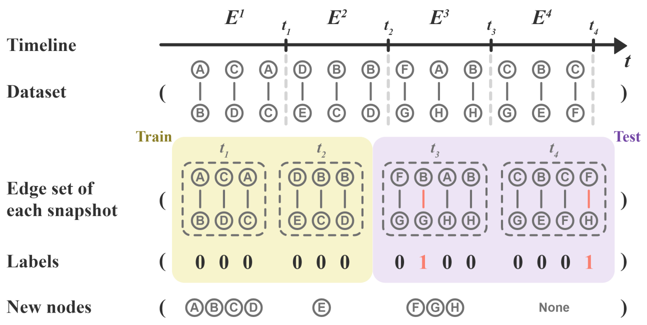

Fig. 1. Data processing for edge anomaly detection on discrete time dynamic graphs: The dataset is aggregated into T snapshots. A model learns to represent edges based on the real edges of the first T_{train} snapshots and predicts the anomaly score of each edge in the T_{test} snapshots. New nodes may appear, so the test phase will have nodes that were not seen in training (nodes F, G and H).

2.1 Representation Learning in Discrete Time Dynamic Graphs

Representation learning consists in projecting input data into a continuous vector space (through an encoder) used for solving downstream tasks (through a decoder) [11]. When input data are static graphs, common encoders include spatial convolution [10,16], spectral convolution [6], Random Walk-based algorithms [9,28] and graph Transformers [24,39,14] (see appendix A.1).

To take into account the time factor involved in dynamic graph representation learning, several approaches [21,42] have been proposed. Most of them choose to process snapshots sequentially [38], using a static node-based graph encoder f_G such as GNN and then use a temporal encoder f_T such as RNN to encode the information computed from f_G across the time steps.

This common paradigm for DTDG encoding, although intuitive and simple to practice, struggles to deal with the apparition of new nodes, as illustrated in figure 1. Hence, since the number of nodes at each time step is not fixed, the input data to f_T will have gaps in the time dimension (see Fig. 2 (a)).

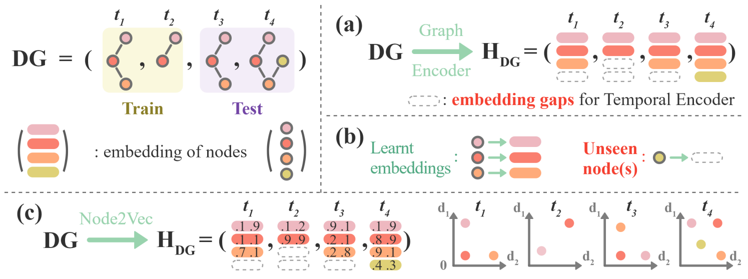

Fig. 2. Three challenges in the coding of discrete-time dynamic graphs: (a) Variation in the number of nodes can lead to the unavailability of a temporal encoder because of the gaps in its inputs. (b) Initial embedding of unseen nodes cannot be learnt. (c) Random Walk-based embedding, such as Node2Vec, is misaligned across snapshots. For example, although the structure of the pink, red, and orange nodes is similar in t_1, t_3, and t_4, their embedding coordinates in t_3 are approximated to those in t_1 but after swapping the coordinates d_1 and d_2, their embedding coordinates in t_4 are approximated to those in t_1 but by a 180°rotation along the point (0.5,0.5).

Another common challenge in real-world applications is the lack of node attributes. Without these attributes, most existing Dynamic Graph Neural Network models first compute initial node features (i.e. an initial node embedding) [35,15], which are used as input to GNN layers or directly as input to the RNN model. These features are usually computed using a model that has been trained to project one-hot node indices into a vectorial space, using some supervised labels provided either at the node level or at the graph level. As the node set evolves, such models generally become unusable as the input dimensions change, as shown in Fig. 2 (b).

To address the above challenges, random walk methods [9,28] are often used. Such methods rely only on the graph structure to obtain node representations and are thus feasible even on unattributed graphs.

Definition 3 (Random Walk).
A Random Walk of length L on a graph $G = (V, E)$, where V and E denote the sets of nodes and edges, is a sequence of vertices $\langle v_1, v_2, \ldots, v_L \rangle$ such that each v_{i+1} is chosen randomly from the neighbours of v_i.

Inspired by Word2Vec [23,8], Random Walk-based approaches directly exploit the structural information to compute the node embedding, satisfying that nodes which often appear in the same walk have similar embeddings and vice versa. In simple terms, such methods learn by attempting to predict a node correctly based on the contextual nodes [9,28,25]. For example, for a random walk $\langle v_a, v_b, v_c, v_d, v_e, v_f, v_g, ... \rangle$ (Def. 3), by hiding the node v_d, the model attempts to find the most appropriate node for the contextual nodes $\langle v_b, v_c, ?, v_e, v_f \rangle$. These methods thus find the node embedding in a self-supervised manner.

While efficient at clustering similar nodes and distancing dissimilar ones in a static context, Random Walk methods struggle with alignment in the embedding space across multiple snapshots for structurally identical or similar graphs [31]. This is due to the stochastic nature of the algorithm and its focus on optimizing relative distances between nodes based on their connectivity patterns. As a consequence, such methods cannot ensure that nodes with similar roles in different graphs are mapped to close points in vector space, directly affecting their availability for DTDGs. An example is shown in Fig. 2 (c).

Despite the fact that several approaches have emerged for DTDG representation learning, such as the "Sequentially Encoding" paradigm [38], and methods based on "Node Index Embedding" or Random Walk, DTDG inductive learning remains an open problem. A solution to this problem is crucial, especially given that in practice, anomaly detection in social networks and trading systems needs to be able to react to newly added nodes.

2.2 Anomaly Detection in Dynamic Graphs

Due to the scarcity of anomaly data, "fully supervised anomaly detection is often impractical", as stated in the survey in the field of Deep Learning Anomaly Detection [27]. Instead, anomaly detection is tackled using either one-class strategies (one-class SVM [30], SVDD [32]), which uses only positive data to model the regularity, either the generation of synthetic anomalies for casting anomaly detection into a binary classification problem. In practice, the latter strategy is widely adopted since it is more suitable for discriminant models such as neural networks. The methods used to detect edge anomalies in DTDGs [42,4,19] rely on the generation of synthetic anomalies associated with neural networks. A model learns the representation from real edges and synthetic anomalous edges from the first T_{train} snapshots. The remaining T_{test} snapshots are reserved for testing the model's performance.

To address the anomaly detection problem in DTDGs, the early model AddGraph [42] proposed to use GCN [16] as a static graph encoder followed by a GRU [5] as a temporal encoder. However, this approach requires the entire node set as input and cannot effectively handle newly added nodes [13], as mentioned in section 2.1. Focusing on the structure of the target edge, StrGNN [4] takes as input the h-hop enclosing subgraphs of the latest τ time steps of each target edge. It reserves the most relevant K nodes by Sortpooling [41], then encodes the node embedding matrix of shape $K \times d$ across τ snapshots by GRU [5], and apply a readout function to obtain the edge representation.

The state-of-the-art approach TADDY [19] is also based on subgraph encoding, but makes many improvements in its implementation. Specifically, it computes the most related K nodes of an edge by the Personalized PageRank (PPR) [26]. Through this operation in the τ latest snapshots, a spatio-temporal subgraph is then formed with a total of $\tau \times K$ nodes. For each node in this spatio-temporal subgraph, TADDY uses three encodings based on PPR diffusion, graph distance and relative time encoding. It then propagates the information through a Transformer Encoder structure [34]. Finally, the mean-pooled representation of these $\tau \times K$ nodes is used as the edge representation.

By focusing on subgraphs, both TADDY and StrGNN avoid the difficult problem of computing a node embedding for the whole snapshot, i.e., a graph-level node embedding. However, such a strategy disallows long-range propagation on the snapshot. In particular, StrGNN [4] performs similarly at 1-hop, 2-hop, and 3-hop subgraph scopes. Likewise, the performance of TADDY is not improved when the number of subgraph nodes is greater than 7 and the number of snapshots is greater than 2 [19]. This phenomenon suggests a potential weakness of these models in capturing long-range interactions, which is essential to better understand complex interdependencies between nodes [11,33], especially when several distant subgraphs in the network show similar evolutionary trends.

Besides challenges in capturing long-range topological interactions, recent studies [29] highlight the importance of memory mechanisms for encoding historical information and temporal long-range dependencies. Therefore, efficient dynamic graph encoding must consider both short-range and long-range dependencies in time and space. This article presents an approach that balances information propagation in long-range and local subgraphs in the inductive setting.

3 Proposed Approach

This section introduces our contributions to edge anomaly detection in DTDGs. It first describes our novel method **ACWA** for inductive graph-level node embedding. Then we propose the overall structure of our model **DCIDGT** which addresses global and local propagation problems in both spatial and temporal dimensions. Finally, we introduce the overall inference procedure of the model.

3.1 Accumulative Causal Walk Alignment for inductive embedding

In this subsection, we propose the **Accumulative Causal Walk Alignment (ACWA)** method to address both the problems of RW embedding misalignment and the need for long-range temporal information. Our proposal relies on solving the orthogonal procruste problem for embeddings of accumulative causal walks.

As mentioned before, a difficulty of random walk-based approaches applied on DGs is the alignment of embedding across snapshots. To our knowledge, the only existing studies propose some variants of Orthogonal Procruste to solve this problem [31]. The Orthogonal Procruste problem [12], mathematically defined in Def. 4, seeks an orthogonal matrix $\mathbf{\Omega}$ that most closely maps one set of vectors to another, and is thus theoretically suitable for semantic alignment.

Definition 4 (Orthogonal Procruste). *The Orthogonal Procruste problem solves for* $\mathbf{\Omega}$ *in the equation* $\min_{\mathbf{\Omega}} \|\mathbf{A} - \mathbf{B}\mathbf{\Omega}\|_F$*, subject to* $\mathbf{\Omega}^{\mathbf{T}}\mathbf{\Omega} = \mathbf{I}$*, where* $\mathbf{I}$ *is the identity matrix,* $\|\cdot\|_F$ *is the Frobenius norm,* $\mathbf{A}$ *and* $\mathbf{B}$ *are matrices of the same size. Specifically, in our case,* $\mathbf{A}$ *and* $\mathbf{B}$ *are the embedding matrices of the reference nodes of snapshots at* $t-1$ *and* t *respectively.*

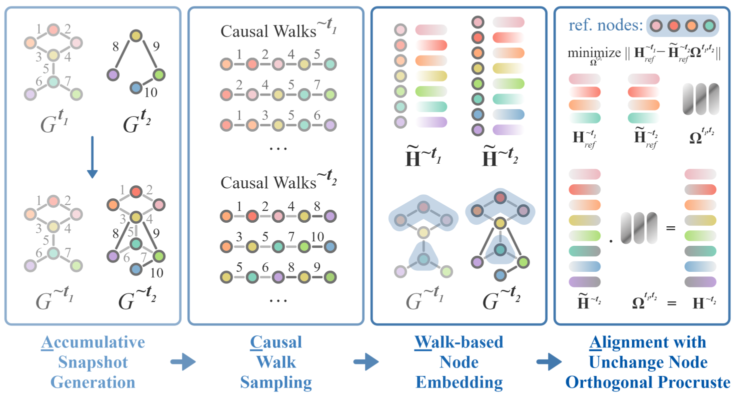
Causal Walks $^{\sim t_1}$
Causal Walks $^{\sim t_2}$
ref. nodes:
minimize$_{\Omega}$ $\| \mathbf{H}^{\sim t_1}_{ref} - \tilde{\mathbf{H}}^{\sim t_2}_{ref} \Omega^{t_1,t_2} \|$
G^{t_1}
G^{t_2}
$G^{\sim t_1}$
$G^{\sim t_2}$
$\tilde{\mathbf{H}}^{\sim t_1}$
$\tilde{\mathbf{H}}^{\sim t_2}$
$\mathbf{H}^{\sim t_2}$
Ω^{t_1,t_2}
Accumulative Snapshot Generation
Causal Walk Sampling
Walk-based Node Embedding
Alignment with Unchange Node Orthogonal Procruste

Fig. 3. The **Accumulative Causal Walk Alignment** approach: 1) For a given moment t, an edge set in chronological order up to moment t is constructed, forming the "Accumulative Snapshot". e.g., accumulative snapshot at t_2, denoted as $G^{\sim t_2}$, contains all edges from moment t_1 and t_2. 2) Random walks considering causality are sampled on each accumulative snapshot. 3) Node-level primitive embedding $\tilde{\mathbf{H}}$ for each accumulative snapshot can be obtained based on causal random walks, e.g., $\tilde{\mathbf{H}}^{\sim t_1}$ and $\tilde{\mathbf{H}}^{\sim t_2}$. The nodes in the two snapshots whose structure has not changed, those shaded in blue in the figure, will be used as the reference node set for alignment. 4) Alignment process: For instance with $G^{\sim t_1}$ as the reference, thus its graph-level node embedding equals to its primitive embedding: $\mathbf{H}^{\sim t_1} = \tilde{\mathbf{H}}^{\sim t_1}$. A linear transformation $\mathbf{\Omega}^{t_1,t_2}$ is found by Orthogonal Procruste (OP) to minimise the embedding of these reference nodes $\mathbf{H}^{\sim t_1}_{ref}$ and $\tilde{\mathbf{H}}^{\sim t_2}_{ref}$ between the two accumulative snapshots. The final globally available node embedding for $G^{\sim t_2}$ is obtained through the OP transformation of its primitive embedding: $\mathbf{H}^{\sim t_2} = \tilde{\mathbf{H}}^{\sim t_2}\mathbf{\Omega}^{t_1,t_2}$.

Intuitively, if a node has no structural changes between two adjacent snapshots, its embedding should also remain almost identical. These nodes are thus considered reference nodes for alignment purposes [31]. However, following the predecessors' task modelling, one cannot know the structurally invariant nodes at t without using historical information. An example is shown in Fig. 1 where a snapshot at t is represented by a list of edges occurring at time step t.

As the foundation to overcome this problem, we propose the "Accumulative Snapshot", which accumulates all previous edges. In particular, an edge set in chronological order up to moment t is constructed, forming the "Accumulative Snapshot" at t, denoted as $G^{\sim t}$. As shown in Fig. 3 (1), accumulative snapshot at t_2, denoted as $G^{\sim t_2}$, contains all edges in time steps t_1 and t_2.

Meanwhile, RW methods have the shortcoming of focusing only on structural information, but not on temporal information. Therefore, random walks considering causality have been proposed in recent studies [25,36], as defined in Def. 5. Drawing inspiration from this, we consider the chronological order as causality, sample Causal Walks from Accumulative Snapshots, as shown in Fig. 3 (2), and compute the primitive embeddings $\tilde{\mathbf{H}}^{\sim t}$ and reference node set. Owing to the "Accumulative Snapshot", all seen nodes that are not currently participating in the edges can be used as references and learn the embedding under the current structure, which provides a more solid reference node set for orthogonal procruste, see Fig. 3 (3).

Definition 5 (Causal Random Walk). *A Causal Random Walk, (simplified as "Causal Walk"), is a random walk $\langle v_1, v_2, \ldots, v_L \rangle$ such that the timestamp of the edge between nodes v_i and v_{i+1} is less than or equals to the timestamp of the edge between nodes v_{i+1} and v_{i+2}, denoted as $t_{e_{v_i v_{i+1}}} \leq t_{e_{v_{i+1}, v_{i+2}}}$.*

Finally, using a particular accumulative snapshot as a reference snapshot, e.g., $G^{\sim t_1}$, the initial embedding of that snapshot will be used as the final embedding, i.e., $\mathbf{H}^{\sim t_1} = \tilde{\mathbf{H}}^{\sim t_1}$. All the nodes that remain structurally unchanged in it and its neighbouring snapshot ($G^{\sim t_2}$), are used to find a matrix $\mathbf{\Omega}^{t_1, t_2}$ with Orthogonal Procruste (OP) by minimising the embedding of these reference nodes, see Eq. 1. The final graph-level node embedding for $G^{\sim t_2}$ is obtained through the OP transformation of its primitive embedding, as shown in Fig. 3 (4) and Eq. 2. By chaining this operation, $\mathbf{\Omega}^{t,t+1}$ can be found based on $\mathbf{H}^{\sim t}_{ref}$ and $\tilde{\mathbf{H}}^{\sim t+1}_{ref}$ such that nodes in all snapshots are embedded in the same space.

$$\min_{\mathbf{\Omega}^{t_1, t_2}} \|\mathbf{H}^{\sim t_1}_{ref} - \tilde{\mathbf{H}}^{\sim t_2}_{ref} \mathbf{\Omega}^{t_1, t_2}\|_F \tag{1}$$

$$\mathbf{H}^{\sim t_2} = \tilde{\mathbf{H}}^{\sim t_2} \mathbf{\Omega}^{t_1, t_2} \tag{2}$$

The ACWA method, drawing inspiration from causal walk and incorporating orthogonal procruste between snapshots, allows for graph-level node embedding over longer paths considering all historical information. This lays the foundation for capturing long-range interaction which will be presented in the next section.

3.2 Dual-Contextual Inductive Dynamic Graph Transformer

The graph-level node embedding from ACWA enables long-range propagation of the spatio-temporal information. In this subsection, we present the overall structure of our model: **Dual-Contextual Inductive Dynamic Graph Transformer (DCIDGT)**. It employs two Transformer modules for graph-level (global context) and subgraph-level (local context) encoding respectively. Their concatenated outputs form the final edge embedding, as shown in Fig. 4.

Global Contextual Transformer As a module for propagating long-range information, compared to the cubic complexity of spectral convolution [2], Transformers has the advantages of a quadratic complexity and the ability to capture long-range interactions.

Once the ACWA graph-level node embedding computed, such embedding can be used simultaneously as node representation and positional encoding. Information propagation on a snapshot is then implemented through a Global Contextual Transformer (GCT). The input of this Transformer is the d-dimensional

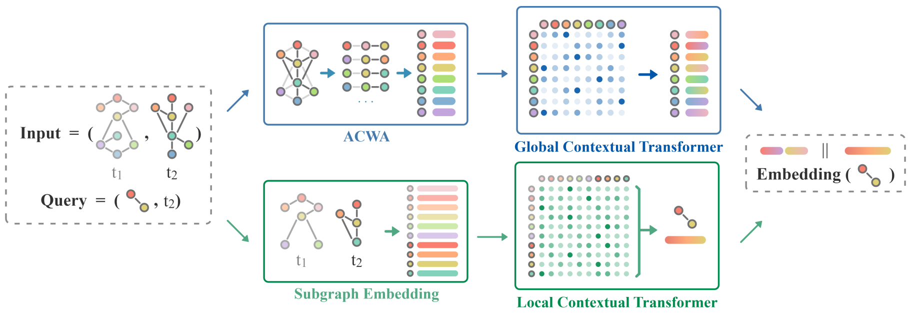

Fig. 4. Overall Architect of Dual-Contextual Inductive Dynamic Graph Transformer (DCIDGT): Left: The graph-level node embedding is computed by the Accumulative Causal Walk Alignment module, then sent to a Transformer for global context encoding. Right: The subgraph-level node embedding is computed by the Subgraph Embedding module, and then encoded by a Local Contextual Transformer. The concatenation of these embeddings serves as the edge embedding for downstream tasks.

embedding of N^t nodes of the snapshot G^t, denoted as $\mathbf{H}^t_{global} \in \mathbb{R}^{N_t \times d}$. The Transformer Encoder maps $\mathbf{H}^t_{global}$ to Query, Key, and Value matrices through three learnable weight matrices $\mathbf{W_Q}$, $\mathbf{W_K}$, and $\mathbf{W_V}$, see Eq. 3.

$$\mathbf{Q}^t = \mathbf{H}^t_{global}\mathbf{W}^{\mathbf{Q}}, \quad \mathbf{K}^t = \mathbf{H}^t_{global}\mathbf{W}^{\mathbf{K}}, \quad \mathbf{V}^t = \mathbf{H}^t_{global}\mathbf{W}^{\mathbf{V}} \tag{3}$$

$$\mathbf{Z}^t_{global} = \text{Attention}(\mathbf{Q}^t, \mathbf{K}^t, \mathbf{V}^t) = \text{softmax}\left(\frac{\mathbf{Q}^t \mathbf{K}^{t^T}}{\sqrt{d}}\right)\mathbf{V}^t \tag{4}$$

Then, the similarity between each query and all the keys is computed, scaled, and normalised to obtain the attention weights, see Eq. 4. This self-attention matrix can be represented as in Fig. 4 (left). Through self-attention, the representation $\mathbf{z}^t_{i,global}$ of each node i is dynamically adapted based on the relationship with other nodes by aggregating information from the entire snapshot.

Local Contextual Transformer To propagate local information, node representations need to be valid only within the subgraph. To this end, we adopt the TADDY model [19] as a key part of our Local Contextual Transformer.

As presented in section 2, for each target edge, a subgraph is formed by the $\tau \times K$ most relevant nodes computed by Personalised PageRank (PPR) (see appendix. A.2). Each node in the subgraph is then embedded using the PPR diffusion-based, the shortest path distance-based, and the relative time-based encodings. These three encodings are summed up to subgraph-level node embedding $\mathbf{h}^{t'}_{i,local}$ using Eq. 5. These components are detailed in appendix. A.3.

$$\mathbf{h}^{t'}_{i,local} = \mathbf{W}_1(rank(\mathbf{s}^{t'}_i[idx(v^{t'}_i)])) + \mathbf{W}_2(dist(v^{t'}_i, e^{t'}_{target})) + \mathbf{W}_3(\|t - t'\|) \tag{5}$$

The Local Contextual Transformer (LCT) has the same structure as the GCT. It takes the subgraph-level node embeddings $\mathbf{H}^t_{local} \in \mathbb{R}^{\tau K \times d}$ as input. A pooler averages the output subgraph representation as the edge embedding.

Finally, the concatenation of the outputs of the two Transformers serves as the edge embedding, as shown in Fig. 4.

3.3 Overall Inference Procedure

The overall inference procedure of the model DCIDGT is depicted in Algo. A.4. For the input snapshot G^t and its history information, ACWA and Subgraph Embedding modules obtain the graph-level and subgraph-level node embeddings, respectively. Then, the Global Contextual Transformer computes the graph-level node representation $\mathbf{Z}^t_{global} \in N^t \times d$. For each of the M^t edges, the Local Contextual Transformer computes subgraph-level node representation of shape $\tau K \times d$, and pooled to $1 \times d$.

Finally, for an edge e^t_m consists of nodes v_i and v_j, the outputs from these three modules, $\mathbf{z}^t_{i,global}$, $\mathbf{z}^t_{j,global}$ and $\mathbf{z}^t_{m,local}$ are concatenated as the final embedding of the edge, as shown in Fig. 4. A linear projection layer then outputs the anomaly score based on this embedding.

4 Experimental evaluation

4.1 Datasets and Baselines

For evaluating the performance of the model, experiments are conducted on the publicly available cryptocurrency transactions datasets: Bitcoin-Alpha [18] and Bitcoin-OTC [18]. The task is to predict the anomaly score of the edges in a snapshot, given the ordered edge lists from the previous snapshot(s) as input. The statistics of the datasets are presented in table 1. The Bitcoin datasets capture the interactions of Bitcoin users in an anonymous transaction network. Nodes model users while edges represent trustworthiness ratings between users.

Table 1. Dataset Statistics. The numbers of nodes and edges are based on the original datasets. The numbers of non-duplicated edges and unseen nodes (in the training phase) are statistics after pre-processing based on the protocols in preceding studies [4,19].

Name	# Nodes	# Edges	# Non-duplicated Edges	# Unseen Nodes
Bitcoin-Alpha	3,783	24,173	14,124	998 (26%)
Bitcoin-OTC	5,881	35,588	21,492	2,218 (38%)

We compare our model against the state-of-the-art TADDY model and other influential models for the dynamic graph anomaly detection task. Since DCIDGT fully retains TADDY's subgraph embedding and Local Contextual Transformer, TADDY can be viewed as an ablation study variant of DCIDGT without ACWA and GCT. In addition to the models **AddGraph** [42], **StrGNN** [4], and **TADDY** [19] presented in section 2, we also include **NetWalk** [40] in the baseline models. NetWalk [40] first uses an autoencoder to learn embeddings from random walks. Then the model updates node representations and sampled walks when new edges emerge. The anomaly score of an edge is then computed through the Hadamard product of the embeddings of two nodes.

Each of these models offers insights into the evolving landscape of anomaly detection in dynamic graphs, serving as a solid foundation upon which our proposed method, DCIDGT, seeks to innovate.

4.2 Experimental settings

To ensure the fairness of our experiments, we follow exactly the experimental settings and snapshot splitting protocol outlined in TADDY [19], and experimented on the two bitcoin datasets referred to in its code[4]. This section describes the general settings, while appendix A.5 introduces the hyperparameters and other details of the experiments.

The dataset is segmented into two subsets: the initial 50% of snapshots form the training set, while the subsequent 50% of snapshots constitute the test set. The data were processed in the same way as in [19], i.e., after removing duplicate edges, the Bitcoin-OTC and Bitcoin-Alpha datasets have been aggregated to 1,000 and 2,000 edges per snapshot, respectively.

The experimental framework sets up the anomaly detection task with three different percentages of anomalies p_A: 1%, 5%, and 10%, and randomly injects anomalous edges in each test snapshot. Following the settings of the baseline models, the "Area Under the Receiver Operating Characteristic Curve" (AUC ROC) serves as a metric. However, the "Area Under the Precision-Recall Curve" (PR AUC) is considered more suitable for unbalanced datasets and is often used for anomaly detection tasks [7]. Therefore, we use both metrics for the evaluation. They range from 0 to 1, with higher values indicating better performance.

4.3 Experiment results

The results for TADDY and our DCIDGT model are averaged over five runs with seeds 1-5 for the generation of anomalous edges and the initialisation of model parameters. AUC ROC Results for all the baseline methods are taken directly from their publications [42,40,4,19]. Given the space constraints, the standard deviations of experiments are reported in A.6.

Tables 2 and 3 show the results for each anomaly percentage configuration of the Bitcoin datasets. Regarding the PR AUC metric, which is more appropriate for anomaly detection tasks, DCIDGT significantly outperforms TADDY by capturing long-term dependencies. While gaining a higher precision, DCIDGT still achieves a high AUC ROC metric, mostly outperforming TADDY's results.

[4] https://github.com/yuetan031/TADDY_pytorch/tree/main

Table 2. Performance of edge anomaly detection with anomaly percentages $p_A = 1\%, 5\%$, and 10%, showing the mean of the PR AUC values for seeds 1-5, with the best results highlighted in bold.

	Bitcoin-Alpha			Bitcoin-OTC		
Methods	1%	5%	10%	1%	5%	10%
TADDY	0.127	0.417	0.562	0.093	0.334	0.502
DCIDGT (ours)	**0.274**	**0.565**	**0.722**	**0.196**	**0.472**	**0.593**

Table 3. Performance of edge anomaly detection with anomaly percentages $p_A = 1\%, 5\%$, and 10%, showing the mean of the AUC ROC values for seeds 1-5, with the best results highlighted in bold.

	Bitcoin-Alpha			Bitcoin-OTC		
Methods	1%	5%	10%	1%	5%	10%
NetWalk	0.839	0.836	0.835	0.779	0.769	0.753
AddGraph	0.867	0.840	0.850	0.835	0.846	0.859
StrGNN	0.857	0.867	0.863	0.901	0.878	0.884
TADDY	0.945	0.934	0.942	**0.946**	0.934	**0.943**
DCIDGT (ours)	**0.954**	**0.951**	**0.953**	**0.946**	**0.937**	0.936

Despite the performance improvement, DCIDGT does not have a significant increase in computation over TADDY, see the complexity analysis in appendix A.7. Additional experiments in A.8 show that **ACWA** module exhibits stronger representation learning than the commonly used node index embedding, highlighting its potential and effectiveness in dynamic graph representation learning.

5 Conclusion

In this work, we presented the **DCIDGT** model, an architecture for dynamic graph edge anomaly detection that focuses on the integration of both global and local spatio-temporal information. We achieved new state-of-the-art results in most experiments on two publicly available Bitcoin datasets.

Different from existing state-of-the-art algorithms that are limited to subgraph information, **ACWA** proposes a novel solution for embedding and aligning unseen nodes in dynamic graphs. Empirical results demonstrate the potential of our approach, which opens the door to future research on more optimised alignment and random walk sampling, pushing the boundaries of anomaly detection in dynamic graphs. In future research, other alignment methods besides OP could be explored for ACWA. Furthermore, extending ACWA to other dynamic graph encoders and applying it to a broader range of tasks could provide valuable insights.

Acknowledgments. This work was financially supported by the ANR Labcom Lisa ANR-20-LCV1-0009.

Disclosure of Interests. All authors disclosed no relevant relationships.

References

1. Alon, U., Yahav, E.: On the bottleneck of graph neural networks and its practical implications. arXiv preprint arXiv:2006.05205 (2020)
2. Balcilar, M., Guillaume, R., Héroux, P., Gaüzère, B., Adam, S., Honeine, P.: Analyzing the expressive power of graph neural networks in a spectral perspective. In: Proceedings of the International Conference on Learning Representations (ICLR) (2021)
3. Cai, D., Qian, S., Fang, Q., Hu, J., Xu, C.: User cold-start recommendation via inductive heterogeneous graph neural network. ACM Transactions on Information Systems **41**(3), 1–27 (2023)
4. Cai, L., Chen, Z., Luo, C., Gui, J., Ni, J., Li, D., Chen, H.: Structural temporal graph neural networks for anomaly detection in dynamic graphs. In: Proceedings of the 30th ACM international conference on Information & Knowledge Management. pp. 3747–3756 (2021)
5. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
6. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. Advances in neural information processing systems **29** (2016)
7. Gaudreault, J.G., Branco, P., Gama, J.: An analysis of performance metrics for imbalanced classification. In: International Conference on Discovery Science. pp. 67–77. Springer (2021)
8. Goldberg, Y., Levy, O.: word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 (2014)
9. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)
10. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)
11. Hamilton, W.L.: Graph representation learning. Morgan & Claypool Publishers (2020)
12. Hurley, J.R., Cattell, R.B.: The procrustes program: Producing direct rotation to test a hypothesized factor structure. Behavioral science **7**(2), 258 (1962)
13. Kim, H., Lee, B.S., Shin, W.Y., Lim, S.: Graph anomaly detection with graph neural networks: Current status and challenges. IEEE Access **10**, 111820–111829 (2022)
14. Kim, J., Nguyen, D., Min, S., Cho, S., Lee, M., Lee, H., Hong, S.: Pure transformers are powerful graph learners. Advances in Neural Information Processing Systems **35**, 14582–14595 (2022)
15. King, I.J., Huang, H.H.: Euler: Detecting network lateral movement via scalable temporal link prediction. ACM Transactions on Privacy and Security **26**(3), 1–36 (2023)
16. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
17. Kumar, S., Hooi, B., Makhija, D., Kumar, M., Faloutsos, C., Subrahmanian, V.: Rev2: Fraudulent user prediction in rating platforms. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. pp. 333–341. ACM (2018)

18. Kumar, S., Spezzano, F., Subrahmanian, V., Faloutsos, C.: Edge weight prediction in weighted signed networks. In: Data Mining (ICDM), 2016 IEEE 16th International Conference on. pp. 221–230. IEEE (2016)
19. Liu, Y., Pan, S., Wang, Y.G., Xiong, F., Wang, L., Chen, Q., Lee, V.C.: Anomaly detection in dynamic graphs via transformer. IEEE Transactions on Knowledge and Data Engineering (2021)
20. Ma, X., Wu, J., Xue, S., Yang, J., Zhou, C., Sheng, Q.Z., Xiong, H., Akoglu, L.: A comprehensive survey on graph anomaly detection with deep learning. IEEE Transactions on Knowledge and Data Engineering **35**(12), 12012–12038 (2021)
21. Manessi, F., Rozza, A., Manzo, M.: Dynamic graph convolutional networks. Pattern Recognition **97**, 107000 (2020)
22. Meyer, D.: How exactly does word2vec work. Uoregon. Edu, Brocade. Com pp. 1–18 (2016)
23. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
24. Min, E., Chen, R., Bian, Y., Xu, T., Zhao, K., Huang, W., Zhao, P., Huang, J., Ananiadou, S., Rong, Y.: Transformer for graphs: An overview from architecture perspective. arXiv preprint arXiv:2202.08455 (2022)
25. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: Companion proceedings of the the web conference 2018. pp. 969–976 (2018)
26. Page, L., Brin, S., Motwani, R., Winograd, T., et al.: The pagerank citation ranking: Bringing order to the web (1999)
27. Pang, G., Shen, C., Cao, L., Hengel, A.V.D.: Deep learning for anomaly detection: A review. ACM computing surveys (CSUR) **54**(2), 1–38 (2021)
28. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)
29. Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., Bronstein, M.: Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637 (2020)
30. Schölkopf, B., Williamson, R.C., Smola, A., Shawe-Taylor, J., Platt, J.: Support vector method for novelty detection. Advances in neural information processing systems **12** (1999)
31. Tagowski, K., Bielak, P., Kajdanowicz, T.: Embedding alignment methods in dynamic networks. In: International Conference on Computational Science. pp. 599–613. Springer (2021)
32. Tax, D.M., Duin, R.P.: Support vector data description. Machine learning **54**, 45–66 (2004)
33. Topping, J., Di Giovanni, F., Chamberlain, B.P., Dong, X., Bronstein, M.M.: Understanding over-squashing and bottlenecks on graphs via curvature. arXiv preprint arXiv:2111.14522 (2021)
34. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
35. Wang, D., Zhang, Z., Ma, Y., Zhao, T., Jiang, T., Chawla, N.V., Jiang, M.: Learning attribute-structure co-evolutions in dynamic graphs. arXiv preprint arXiv:2007.13004 (2020)
36. Wang, Y., Chang, Y.Y., Liu, Y., Leskovec, J., Li, P.: Inductive representation learning in temporal networks via causal anonymous walks. arXiv preprint arXiv:2101.05974 (2021)

37. Wu, Q., Yang, C., Yan, J.: Towards open-world feature extrapolation: An inductive graph learning approach. Advances in Neural Information Processing Systems **34**, 19435–19447 (2021)
38. Yang, L., Chatelain, C., Adam, S.: Dynamic graph representation learning with neural networks: A survey. IEEE Access pp. 1–1 (2024). https://doi.org/10.1109/ACCESS.2024.3378111
39. Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Liu, T.Y.: Do transformers really perform badly for graph representation? Advances in neural information processing systems **34**, 28877–28888 (2021)
40. Yu, W., Cheng, W., Aggarwal, C.C., Zhang, K., Chen, H., Wang, W.: Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 2672–2681 (2018)
41. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Proceedings of the AAAI conference on artificial intelligence. vol. 32 (2018)
42. Zheng, L., Li, Z., Li, J., Li, Z., Gao, J.: Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In: IJCAI. vol. 3, p. 7 (2019)

A Appendices

A.1 Graph Convolution

A foundational concept of Graph Neural Network (GNN) is Graph Convolution, which can be classified into two main categories: spatial-based and spectral-based [2].

Spatial convolution is analogous to image convolution in that they both focus on local neighbourhoods and can be extended to k-hop neighbourhoods by stacking k layers. However, spatial convolutions [10,16] lead to problems such as over-smoothing and over-squashing [1] and are therefore weaker in capturing the global structure.

On the other hand, spectral convolutions [6] are grounded in spectral graph theory. They transform the graph signal to the spectral domain and convert it back to the original space after filtering it with learnable parameters. Although spectral convolution can capture global structure, the projection in the spectral domain is computationally expensive (of cubic complexity) [2].

A.2 Personalised PageRank

Personalised PageRank (PPR) is an algorithm for finding the nodes in a graph that are most relevant to a given node. It can be computed using the equation 6 and represented by the diffusion matrix $\mathbf{S}$, where the i-th row $\mathbf{s}_i$ represents the connectivity of the i-th node to all other nodes. The matrix $\mathbf{I}_n$ is the identity matrix, $\mathbf{A}$ is the adjacency matrix, $\mathbf{D}$ is the degree matrix, and α is a hyperparameter for the teleport probability.

$$\mathbf{S}^{PPR} = \alpha(\mathbf{I}_n - (1-\alpha)\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})^{-1} \tag{6}$$

A.3 TADDY Model Details

In the TADDY model [19], for each snapshot, a PPR diffusion matrix $\mathbf{S}^t$ is first computed. For the edges $e^t_{a,b}$ consisting of nodes a, b at moment t, the vectors $\mathbf{s}[a]$ and $\mathbf{s}[b]$ in the PPR diffusion matrix are summed up, denoted as $\mathbf{s}_{edge}$. The K n odes having the largest values in $\mathbf{s}_{edge}$ are selected as relevant nodes of the edge $e^t_{a,b}$. Then, for each of the time step $t' \in [t-\tau+1, t]$, the same following operations is performed:

For each node i of the K relevant nodes at moment t', three statistical values are calculated: (1) the value of the ranking of node i $\mathbf{s}_{edge}$; (2) the smaller value of the shortest path from node i to node a or node b; (3) the difference between the moment t' and the target moment t. Each of these three values is embedded through a linear layer, denoted as $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_3$. The output vectors are called PPR diffusion-based encoding, shortest path distance-based encoding, and relative time-based encoding, respectively, as shown in equation 5. They are summed and used as subgraph-level node embeddings for input to the Transformer Encoder. Thus for each $e^t_{a,b}$, the input to this Transformer is the encoding of K nodes for a total of τ time steps.

A.4 Algorithm of DCIDGT

Algorithm 1 Inference Procedure of the DCIDGT Model for Snapshot G^t

```
1: Input: Snapshots until time $t$: $\mathbb{G}^t = \{G^1, \ldots, G^t\}$
2: Output: Anomaly Score $score^t_m$ for each edge $m$ in $G^t$
3: $\mathbf{H}^t_{global} \in \mathbb{R}^{N^t \times d/2} \leftarrow$ ACWA($\mathbb{G}^t$)
4: $\mathbf{Z}^t_{global} \in \mathbb{R}^{N^t \times d/2} \leftarrow$ Global Transformer($\mathbf{H}^t_{global}$)
5: $\mathbf{H}^t_{local} \in \mathbb{R}^{M^t \times K \times \tau \times d} \leftarrow$ Subgraph Embedding($\mathbb{G}^t$)
6: For $m = 1$ to $M^t$ do
7:     Let edge $e^t_m$ connect nodes $v_i$ and $v_j$
8:     $\mathbf{z}^t_{m,local} \in \mathbb{R}^d \leftarrow$ Pooler(Local Transformer($\mathbf{h}^t_{m,local}$)), $\mathbf{h}^t_{m,local} \in \mathbb{R}^{K \times \tau \times d}$
9:     $\mathbf{z}^t_m \leftarrow$ Concatenate($\mathbf{z}^t_{i,global}, \mathbf{z}^t_{j,global}, \mathbf{z}^t_{m,local}$)
10:    $score^t_m \leftarrow \mathbf{z}^t_m \mathbf{W}_s + \mathbf{b}_s$
11: end for
```

A.5 Parameter Settings

For results on the bitcoin datasets, we used exactly the same model structure, as presented below:

For the hyperparameters of the two Transformer Encoders, we use a relatively small number of parameters, namely number of attention layers = 2 and number of attention heads = 2. The embedding dimension d is set to 32 for the Local

Transformer and $d = 16$ for the Global Transformer. For the subgraph embedding module, we retain the settings in TADDY with $K = 5$, $\tau = 2$.

For sampling and training in ACWA, we follow the approach in [25] and set the walk length $L = 30$ and the number of walks per node $= 200$. For Global Contextual Transformer and Local Contextual Transformer, d is set to 32, so the concatenated edge embedding size is $d_{edge} = 64$.

By strictly following the data splitting in TADDY, the total number of snapshots T, for Bitcoin Alpha and Bitcoin OTC are 8 and 22, respectively. Due to anomalous edges injected in a chaotic order, the total number of edges increases, and the number of snapshots may increase by 1.

All models were trained for 100 epochs using the Adam optimiser with a constant learning rate 1e-5. The weight decay for the Transformer models is set to 5e-4. The validation ratio is set to 10% of the training edges and the patience of early stopping is set to 10 epochs.

For the Orthogonal Procruste alignment, the primitive embedding of the last snapshot in the training set was chosen as the reference, with its front and back neighbouring snapshots aligned towards it by chaining.

A.6 Standard Deviation of Experiments

Table 4. Standard Deviation of the anomaly edge detection task.

	AUC ROC						PR AUC					
	Bitcoin Alpha			Bitcoin OTC			Bitcoin Alpha			Bitcoin OTC		
Anomaly%	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%
DCIDGT	0.007	0.004	0.005	0.008	0.005	0.005	0.095	0.078	0.028	0.050	0.029	0.022
TADDY	0.007	0.005	0.008	0.009	0.005	0.006	0.019	0.041	0.044	0.007	0.011	0.012

A.7 Complexity Analysis

The complexity of DCIDGT can be divided into two parts: preprocessing and inference. For a given snapshot, preprocessing needs to be executed only once, while forward propagation is executed once per epoch during training and only once during testing. Preprocessing includes the Accumulative Causal Walk Alignment (ACWA) for the Global Contextual Transformer (GCT) and Subgraph Embedding for the Local Contextual Transformer (LCT). The main complexity of inference comes from GCT (our proposed global transformer) and LCT (the main structure of TADDY).

Preprocessing Complexity As a preprocessing step, the ACWA algorithm essentially acts as an optimized Node2Vec for DTDG with linear complexity [9]. Specifically, this algorithm generates random walks for all nodes up to time t for each snapshot, followed by Word2Vec [23] training method implemented with Gensim [5]. The complexity of generating random walks is $O(\tilde{N}^t \times R \times L)$, where $\tilde{N}^t$ denotes the number of all seen nodes up to time t, R represents the number of random walks, and L represents the length of each random walk. The training complexity of Word2Vec is $O(n \times d)$ [22], where n denotes the total vocabulary size of the corpus, i.e., the cumulative number of nodes $\tilde{N}^t$, and d is the dimension of the generated vectors. In practice, generating random walks is performed only once and the time can be considered negligible, and the word2vec training complexity is linear.

In our experiments, conducted on Google Colab using an Intel Xeon CPU with 13GB of RAM, we set the hyperparameters as follows: $d = 16$, $L = 30$, $R = 200$, and the number of Word2Vec training $epochs = 100$. The preprocessing time per node per snapshot was 0.141 seconds on average for the bitcoin-alpha dataset and 0.145 seconds on average for the bitcoin-OTC dataset.

Subgraph Embedding, as proposed in TADDY [19], primarily derives its complexity from spectral decomposition used for selecting subgraph nodes, resulting in at least $O(\tilde{N^t}^3)$ complexity, which is cubic. Although this preprocessing is less scalable, the number of nodes per snapshot in the bitcoin datasets ranges from 10^2 to 10^3. Thus, the total preprocessing time is not excessive. Using an Intel Xeon CPU, the preprocessing for all snapshots of a single dataset can be completed in under 5 minutes.

Inference Complexity Both GCT and LCT utilise two-layer, two-head Transformer Encoder layers [34]. The main complexity of these layers arises from the self-attention mechanism and the multi-layer perceptron. The computational load of the self-attention mechanism is $3 \times n \times d^2 + 2 \times n^2 \times d$ per head, and the complexity of the two-layer perceptron in the Transformer is $O(n \times d^2)$. Here, n represents the number of input elements to the Transformer, and d represents the embedding dimension.

In our GCT, global nodes need to be processed only once, so n_{GCT} corresponds to N^t. In LCT, the Transformer Encoder computation needs to be performed for each edge, making its complexity M^t times higher. Correspondingly, n in this case corresponds to the number of nodes in the subgraph, i.e., the number of windows times the number of nodes per snapshot, which according to TADDY's hyperparameters is $n_{\mathrm{LCT}} = 7 \times 2 = 14$.

In summary, the inference complexity of GCT arises from its quadratic complexity with respect to N^t, while LCT, although having a smaller number of subgraph nodes, derives its complexity from being executed M^t times. For references, the average N^t for the Bitcoin-Alpha dataset is 845 and $M^t = 2000$, and the average N^t for the Bitcoin-OTC is 595 with $M^t = 1000$. It can be seen

[5] https://radimrehurek.com/gensim/models/word2vec.html

that the ACWA and GCT modules improve the performance of TADDY without significant additional time complexity.

A.8 Effectiveness of ACWA in learning unseen nodes

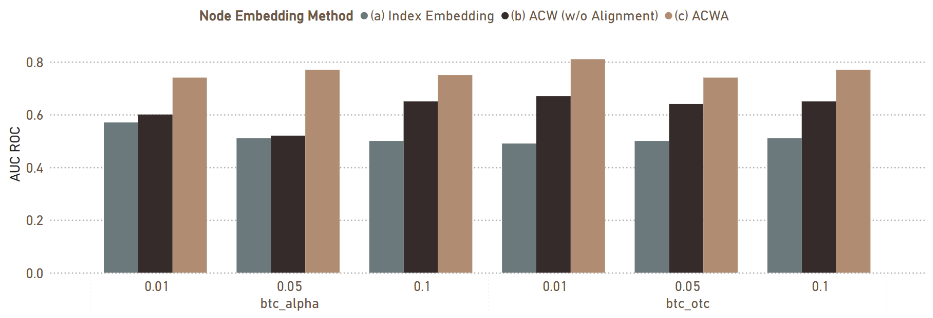

Fig. 5. Edge anomaly detection AUC ROC values for edges consisting only of nodes not seen during training.

In order to better capture the dynamics of the graph, recent studies [15] started to focus on the classification task for unseen edges. Inspired by this, we propose an additional experiment to evaluate the expressiveness of ACWA for globally available unseen node embedding.

Specifically, instead of basing the AUC ROC on all edges, in this experiment, it is computed only on edges consisting entirely of nodes that were not seen during training. This is a more difficult problem due to the complete lack of information about these nodes in the training data, e.g., edges $e^{t_3}_{F,G}$ and $e^{t_4}_{F,H}$ in Fig. 1. To avoid the influence of the model structure, we propose to use a simple structure with only a linear layer with learnable parameters $\mathbf{W}_s$ and $\mathbf{b}_s$ for anomalous score prediction, denoted as $score^t_{i,j} = [\mathbf{z}^t_i \| \mathbf{z}^t_j]\mathbf{W}_s + \mathbf{b}_s$. For comparison, we also examined the commonly used Index Embedding and Accumulative Random Walks without alignment as node embedding methods.

The mean of AUC ROC value under seeds 1-3 is shown in Fig. 5. Index Embedding achieves a score of around 0.5 as it is based on random initialisation of the predictions, which is in line with our expectations. The unaligned ACW method performs at similar levels on the digg and UCI datasets but performs better on the bitcoin datasets. This may be due to the fact that, despite the lack of alignment, it has a certain expressive capacity. ACWA generally outperforms the unaligned ACW, showing its viability for unseen node embedding.