Boosting the performance of Deployable Timestamped Directed GNNs via Time-Relaxed Sampling

Arihant Jain[1]✉, Gundeep Arora[1], and Anoop Saladi[1]

International Machine Learning, Amazon, Bengaluru, India
{arihanta,gundeepa,saladias}@amazon.com

Abstract. Timestamped graphs find applications in critical business problems like user classification, fraud detection, etc. This is due to the inherent nature of the data generation process, in which relationships between nodes are observed at defined timestamps. Deployment-focused GNN models should be trained on point-in-time information about node features and neighborhood, similar to the data ingestion process. However, this is not reflected in benchmark directed node classification datasets, where performance is typically reported on undirected versions of graphs that ignore these timestamps. Constraining the leading approaches trained on undirected graphs to timestamp-based message passing at test time leads to sharp drops in performance. This is driven by the blocking of pathways for neighborhood information, which was available during the undirected training phase but not during the test time, highlighting the label leakage issue in applied graph use-cases. We bridge this mismatch of message passing semantics in directed graphs by first resetting baselines while highlighting the semantic case where undirected training/inference would fail. Second, we introduce TRD-GNN, which bridges performance drop, by leveraging a novel GNN sampling layer that relaxes the time-directed nature of the graph only to the extent that it limits any possibility of labels being leaked during the training phase. The two contributions combined form a recipe for robust GNN model deployment in industry use-cases. Finally, we demonstrate the benefits of the proposed relaxation by drawing out qualitative analysis where it helped improve performance on the node classification task on multiple public benchmark and proprietary e-commerce datasets.

Keywords: Graph Neural Networks · Timestamped directed graphs · Label leakage.

1 Introduction

With the proliferation of e-commerce and social networking services, building and mining relationships between entities on respective services have been useful in improving the customer experience. Social networks (such as Reddit or Facebook) leverage interaction and follow behaviour data, formulated as a graph

to suggest new posts to customers and other users to follow. At the same time, e-commerce stores use user-item graphs for recommendation systems, both helping in improving customer experience and engagement. At the same time, online social communities suffer from abuse [2, 33] where misinformation propagation, trolling, and using offensive language and on e-commerce stores, abuse comprises fraudulent activities such as artificially improving the search ranking of products with fake reviews [10,18]. Such abusive behavior reduces user trust, engagement, and satisfaction. Graph Neural Networks (GNNs) have found applications in such cases where relationships between entities (like users in a social network) can help improve predictability in user engagement tasks like user-follow suggestions. These applications are typically built on top of graphs with timestamped edges, where the timestamp is when the edge is observed.

Citation networks mimic similar behaviour of directed graphs, where published research papers, positioned as nodes on the graph, cite other research papers which were published before the citing paper. The timestamp on the edge is the year when the citing paper is published, and it also forms the basis which training, validation and test instances are divided. This behaviour is manifested in implementation using directed edges which need to be acknowledged in message passing. Consider a case of paper classification where GNNs leverage features of papers cited by the paper to be classified, say p_i to improve its classification accuracy. In this case, in order to maintain train-test parity, training for p_i should leverage features of papers cited by p_i and not those that cite p_i, since they would be published after p_i, even though they may be present in the graph while training. Doing so will lead to model collapse in cases where the paper classification is performed immediately when it is published. [8,19,36] ignore this nuance while setting benchmarks for node classification using their proposed techniques, leaving a loophole for evaluating techniques tailored for timestamped graphs.

This nature of label-leakage is unique to timestamped graphs and can have a detrimental impact when ignored for critical industrial applications. Some of the applied use-cases where this acknowledgement of timestamped edges during training to maintain train-test parity is critical are:

- **User classification**: After the test sample was observed, evaluating user classification models offline that leverage user-user and user-transaction relationships can lead to poor GNN model performance in deployment settings.
- **Fraud Detection**: Using messages passed along edges with fraudulent entities for predicting abuse of another entity leads to improved fraud detection [15, 29]. Incorrect training would leverage fraud label of the node for message passing along edges, even though the label was observed much later.

Business-critical ML applications require point-in-time training of models, and GNN solutions to such applications typically build on top of timestamped graph to ensure messages from relationships occurring in future do not leak label information while training. Not doing this can lead to breakdown of model in production, as labels determining future information will no longer be available. While [21, 24], explored information leakage to adversarial attacks and [17]

aimed at avoiding leakage to node-embeddings in link prediction task, the non-permissible leakage in node classification remains unexplored. To this end, we propose and showcase the following:

- **Time-Relaxed Directed-GNN (TRD-GNN)** : A novel and efficient sampling strategy that acts as a label-leakage-proof approximation of a time-directed graph to undirected graphs, which leads to improved model performance. We publicly released our code here[1].
- **Qualitative Analysis**: Our qualitative analysis shows how TRD-GNN is able to improve over baseline by providing additional same class information.

2 Time Relaxed Directed - GNN

We will refer to a directed graph, $G = (V, E)$ where V are the nodes of the graph, and E are directed edges between nodes, such that $u \rightarrow v$, implies u is connected to v and creation timestamp t of u is less than that of v ($t[u] < t[v]$).

2.1 Graph Neural Networks

Underneath the success of GNNs is the message passing scheme, which aggregates and transforms the representation vectors of neighbors for each node recursively. This message passing is performed over the edges of the sub-graph, which is obtained by recursively sampling the neighborhood of the node to be classified. Formally, let $\mathbf{h}_l(v)$ be the representation of v in the l-th layer of GNN, and then we can update the representation in the $(l+1)$-th layer.

$$\mathbf{h}_{l+1}(v) = \sigma(\mathbf{W}_l \mathbf{AGG}_{u \in \mathcal{N}(v)}(\mathbf{MSG}(\mathbf{h}_l(v); \mathbf{h}_l(u)))) \tag{1}$$

In Eq. 1, $\mathbf{h}_0(v) = X_v$; σ is activation function; $\mathbf{W}_l$ are trainable parameters in the l-th layer; **AGG** denotes aggregator function and **MSG** denotes the message passing function. GCN [16], GraphSAGE [13], GAT [32], GATv2 [3] etc. follow the same framework in Eq. 1.

2.2 Proposed: TRD-GNN

In a typically directed graph, the destination node u, for which the prediction is being made, collects messages from its direct/indirect neighbors but is unaware of other destination nodes that these neighbors point to. Consider an example of directed subgraph as in Fig 1, u, v_1, v_2, v_3 occur with their respective timestamps being $t[u], t[v_1], t[v_2], t[v_3]$ such that $t[u] < t[v_1] < t[v_2] < t[v_3]$. Here, v_2 is unaware of v_1 during message passing. This issue is unique to directed graphs, and one option to bypass this would be to relax the directions in the directed graph during message passing [7]. In directed graphs, if directions are governed by time, relaxing this directional constraint might lead to label leakage leading

[1] https://github.com/amazon-science/trd-gnn

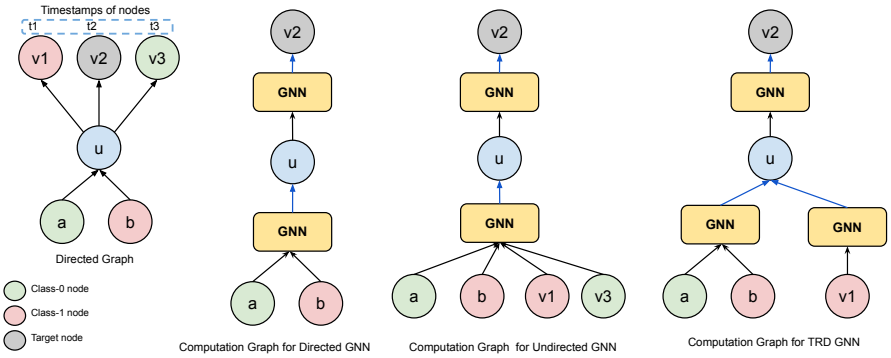

Fig. 1: Figure depicts the computational graphs created by different GNN models (D-GNN, UD-GNN and TRD-GNN) for the given directed graph. It showcases how TRD-GNN acknowledges timestamp ordered-edges by adding new signal from v1 (since t1 < t2) and eliminating signal from v3 (since t2 < t3), leading to improved performance.

to model collapse in production. From here on, we refer to the node's timestamp as its creation timestamp.

To this end, we propose relaxing edge directions for only those nodes placed just before the node on whom the forward pass is done. Following the example, while inferring for v_2, the current setup only samples u. In the case, directions are relaxed without taking in the creation timestamps of nodes, information from v_3 will flow to v_2 via message passing, causing label leakage as v_3 is placed after v_2. However, in our proposed approach, we relax only that sub-graph which contains nodes with timestamps same as or before that of the node being inferred. In our example, since we are inferring for v_2, so for that we relax edges from u to v_1 as v_1 is placed before v_2.

In a vanilla directed graph based message passing, neighborhood for a node u, while accumulating messages for prediction for v, henceforth referred to as *forward* neighborhood (Eq. 2), is defined as:

$$\mathcal{N}^f(u) = \{w | (w \rightarrow u) \in G\} \tag{2}$$

To enable time-relaxed aggregation, we define *backward* neighborhood, $\mathcal{N}_v^b(u)$ in Eq. 3, of a node u, having creation timestamp less than node v, as:

$$\mathcal{N}_v^b(u) = \{w | (u \rightarrow w) \in G, t[w] < t[v]\} \tag{3}$$

The backward neighborhood of u is now dependent on the node v, to whom the message will be further passed on to. The effective neighborhood of u becomes, $\mathcal{N}_v(u) = \mathcal{N}^f(u) \cup \mathcal{N}_v^b(u)$. $\mathcal{N}_v^b(u)$ ensures that messages are accumulated only from those nodes with creation timestamp earlier than v. This differs from leveraging an undirected graph where no time-based filtering would be done while sampling

neighbors. This makes our model susceptible to label leakage leading to poor generalization performance when deployed in production for real-time use-cases.

Just as the total neighborhood of u for destination v is the union of $\mathcal{N}^f(u)$ and $\mathcal{N}^b_v(u)$, message, m^l_{uv}, from source node u to destination node v is defined as a composition of accumulated messages.

$$\begin{aligned} \mathbf{r}^l_{uv}(b) &= \mathbf{AGG}(\{\sigma(\mathbf{W}^l_b \mathbf{h}^0_w) | w \in \mathcal{N}^b_v(u)\}) \\ \mathbf{r}^l_{uv}(f) &= \mathbf{AGG}(\{\sigma(\mathbf{W}^l_f \mathbf{h}^{l-1}_w) | w \in \mathcal{N}^f(u)\}) \\ \mathbf{m}^l_{uv} &= \gamma\left(\sigma(\mathbf{W}^l_f \mathbf{r}^l_{uv}(f)), \sigma(\mathbf{W}^l_b \mathbf{r}^l_{uv}(b))\right) \end{aligned} \tag{4}$$

In Eq. 4, γ is an element-wise function (Combination function) that combine messages to pass from both forward and backward neighbors to create the final message. Once the source node message $\mathbf{m}^l_{uv}$ is computed, it will be passed to the appropriate destination node v. The procedure to get final node embedding is explained in Eq. 4 - 5.

$$\mathbf{h}^l_v = \sigma(\mathbf{W}^l_v \mathbf{AGG}(\{\sigma(\mathbf{W}^l_m \mathbf{m}^l_{uv})\} | u \in \mathcal{N}^f(v))) \tag{5}$$

The layer and the proposed sampling strategy is agnostic to aggregation strategy, be it based on GCN [16], GAT [32], etc. The methodology is extensible to all graph structures as the aggregation strategy can be designed for homogeneous/heterogeneous graphs.

2.3 Adaptation for mini-batch training

Ensuring timestamp-based filtering before message passing setup in directed graphs is non-trivial. For large graphs, we operate in an inductive GraphSAGE [13] batch training setting where for a sample of nodes, the neighborhood is sampled, and messages are aggregated using DGL's [35] message passing API. In the case of TRD-GNN, message $\mathbf{m}^l_{uv}$ can only aggregate messages from nodes w, such that creation time $t[w]$ is earlier than $t[v]$, to ensure that model does not expect future information while inferring for nodes in a real-time production setup. Thus, while sampling a sub-graph from node v, we need to prune edges from nodes with inward edges from neighbors of v based on their creation time. If the nodes in the batch are randomly sampled, then minimum creation time of nodes in the batch can be used to perform the pruning and obtain $\mathcal{N}^b_v(u)$. However, this can result in a sparse subgraph if the difference between the minimum and maximum creation time in the set is large, starving the node with the latest creation time of any meaningful information from $\mathcal{N}^b_v(u)$.

To alleviate this issue, we tweak the sampling strategy of nodes by ensuring that all nodes in a batch are from the same creation-time grain. For example, in a citation network [14], research papers published in the same year or the same conference in a year can be considered together for sampling in a batch. Within the time grain, nodes are sampled randomly. The pseudocode for the proposed mini-batch time relax sampling is present in Algorithm 1.

Algorithm 1: Mini-batch TR Sampling & Embedding computation

Input : Graph G, Set of unique node creation time in graph T, Set of all nodes V
Output: Updated Embedding for nodes in V
for *(t **in** T)* **do**
  $V_t = \{v \mid v \in V, t[v] = t\}$
  $\mathcal{N}^f(V_t) \leftarrow \emptyset$; `/* Create set of source neighbors */`
  for *(v **in** V_t)* **do**
    $\mathcal{N}^f(V_t) \leftarrow \mathcal{N}^f(V_t) \cup \mathcal{N}^f(v)$; `/* Refer to Eq. 2 */`
  $w = \text{random}(V_t)$; `/* Take a random node */`
  $\mathcal{N}^f \leftarrow \emptyset$; `/* Create set of forward neighbors */`
  for *(u **in** $\mathcal{N}^f(V_t)$)* **do**
    $\mathcal{N}^f \leftarrow \mathcal{N}^f \cup \mathcal{N}^f(u)$; `/* Refer to Eq. 2 */`
  $\mathcal{N}^b \leftarrow \emptyset$; `/* Create set of backward neighbors */`
  for *(u **in** $\mathcal{N}^f(V_t)$)* **do**
    $\mathcal{N}^b \leftarrow \mathcal{N}^b \cup \mathcal{N}^b_w(u)$; `/* Refer to Eq. 3 */`
  $G_t = G(V_t, \mathcal{N}^f(V_t), \mathcal{N}^f, \mathcal{N}^b)$; `/* Extract subgraph with given node sets */`
  Update embedding for nodes in V_t using Graph G_t via some message passing API
  ; `/* Refer Eq. 4 - 5 for updating */`

2.4 Comparison with Temporal Graph Networks

Temporal Graph Networks (TGNs) [27] provide a generic framework for deep learning on dynamic graphs represented as sequences of timed events. Our formulation differs from TGNs in two critical aspects. First, TGNs operate on graphs where the same node's state changes with time due to changes in relationships with other entities over time. Thus the edges are timestamped, but nodes are not, and only edges are added with time. Our formulation presents the case where both nodes and edges are timestamped, and once the node is added, its incoming edges do not change with time. The node does not get updated and only new nodes keep getting added to the graph.

Second, TGNs require multiple graphs for training that depend upon the number of different timestamps present in the data. This increases storage requirement, and I/O cost in training such models. Since, in TGNs, there is different graph for different timestamps, it is not susceptible to label leakage. TRD-GNN can be viewed as an efficient alternative for TGNs, working with the assumptions that the incremental nodes and edges added from t to $t+1$ are small as compared to the overall graph and the node/edge features do not change from t to $t+1$. This nature of timestamped graphs makes the TGN approach less scalable, as is evident from the datasets used in [27] have 10K nodes. In TRD-GNN, this is achieved by having a single graph which is a union of the graph over all timestamps. Directed nature of graph and careful sampling

in TRD-GNN ensures that the undirected nature of graph in Temporal GNN is mimic-ed, though much more efficient in terms of storage and I/O cost, allowing it to scale to large graphs with 10M nodes.

3 Experiments

To investigate the effectiveness of the proposed TRD-GNN model, we critically evaluate its impact along three experiment questions: **EQ1**) How does TRD-GNN perform against the vanilla-directed and undirected sampling? **EQ2**) What information does the additional sampling bring in for node classification? **EQ3**) What is the additional computation overhead for the incremental performance of TRD-GNN?

3.1 Experimental Setup

Datasets For a fair comparison with our approach, we benchmark our approach on two open-source datasets and one proprietary e-commerce abuse detection dataset. The open-source time-based citation dataset includes ogbn-arxiv and ogbn-mag [14]. ogbn-arxiv is a homogeneous timestamped directed citation graph, while ogbn-mag is a heterogeneous graph. The statistics of both graphs are present in Table 1, and both are transductive. Both these datasets are directed in nature by virtue of their time so for fair evaluation, we used both directed and undirected graphs to benchmark our approach.

Table 1: Statistics of both public citation and proprietary e-commerce datasets used for experimentation. The proprietary dataset is heavily subsampled and is used to show the efficacy of TRD-GNN on a real-world use-case.

Dataset	#Nodes-Types	#Nodes	#Edges-Types	#Edges	#Classes
ogbn-arxiv	1	169,343	1	1,166,243	40
ogbn-mag	4	1,939,743	4	21,111,007	349
prop. e-commerce	1	$\sim$ 40M	7	$\sim$ 1B	2

E-commerce dataset: We also report results on an e-commerce dataset where we used anonymized and subsampled data from an e-commerce website. The data is subsampled in a manner so as to be non-reflective of actual production traffic. The data consists of 10M users (100K positive) and around 1B interactions aggregated over an arbitrary 2-month window, akin to [9]. Labels indicate nodes marked for abusive behavior. The dataset summary is present in Table 1. We used an inductive setup for its evaluation.

Implementation Details We implemented the experiments using DGL [35] and PyTorch [25]. We use the same hyperparameters and experimental setting

as mentioned here[2]. We use mini-batches of size 1024 during model training and inference. All the experiments were conducted on a 64-core machine with 488 GB RAM running Linux. We used Weights & Biases [1] to track our experiments. We use `cross-entropy` loss for training all our node classification experiments.

Baselines and TRD-GNN Given the GNN layer-type agnostic nature of our proposed method, we leverage different GNN layers like GCN [16], GAT [32] and recently proposed GATv2 [3] as a backbone for ogbn-arxiv dataset. Since obgn-mag is a heterogeneous graph, we use RGCN [28] as a backbone for this dataset.

Further, we introduce notations by defining GNN variants (e.g., GCN), which are trained on different graphs in the following way:

- **UD-GCN**: a GNN model trained on an undirected graph with GCN layer.
- **D-GCN**: another GNN model trained on the directed graph with GCN layer.
- **TRD-GCN**: a TRD-GNN model which uses GCN as an underlying model and is trained on a directed graph.

GAT, GATv2 and RGCN counterparts of these models can be defined similarly. For the e-commerce dataset, we used GraphSAGE [13] model as the GNN layer and D-GNN, UD-GNN and TRD-GNN sampling schemes. All models are evaluated in two different settings:

- **Offline Evaluation**: This setting is similar to a transductive setting where the structure of the test graph remains fixed and is similar to the kind of evaluations that have been done in prior research. This setting works on the undirected graph ignoring the timestamp of the edge creation.
- **Online Evaluation**: This setting is similar to the inductive setting where nodes and edges are added with time in a streaming format and the evaluation is based on the information available at that timestamp. This evaluation mimics the real-life deployment setup where new data is observed, ingested and predictions are made in real-time.

3.2 Performance Comparison (EQ1)

The objective of TRD-GNN is to maximize the information propagation in directed graphs by reversing edges contextual to the node to be classified while not violating the strict time-directed nature of information flow, i.e. avoiding any form of label leakage. Thus, TRD-GNN lies in the middle of the spectrum, where on one end is the vanilla directed graph where information flows based on edge direction, and the other is the undirected graph where no direction based on time is regarded in message passing. Thus, we expect the incremental information brought in by TRD-GNN to improve over the directed GNN and not break down in online inference as in the case of undirected graph training. In

[2] https://github.com/dmlc/dgl/tree/master/examples/pytorch/ogb/ogbn-arxiv

this sub-section, we compare the performance of three GNN training methodologies using different GNN layers for three different datasets. TRD-GNN operates within the guard rails to avoid extra leakage while still expanding its scope of nodes that participate in message passing, thus resulting in higher performance.

On the ogbn-arxiv citation dataset, Table 2 showcases the improvement of TRD-GNN performance over the directed and undirected message passing graph for GCN, GAT and GATv2 backbones, respectively, in online or production setting. This shows the efficacy of our approach independent of backbone architecture and can be easily adapted with other architectures. Furthermore, the huge drop in the performance of the undirected graph-based model (UD-GNN) by 5% on average (from offline evaluation to online evaluation) is because of the consumption of future information during the training of the GNN model. TRD-GNN prevents this, resulting in higher performance.

It is noteworthy that in the case of TRD-GNN, the model performance during offline evaluation is close enough to model performance during online evaluation. This advantage makes the model consistent and reliable to use in production setting. Furthermore, TRD-GNN shows consistent improvement in both inductive and transductive setting.

Table 2: Table reports the results in terms of accuracy (in %) on the homogeneous ogbn-arxiv citation dataset and the heterogeneous ogbn-mag dataset with multiple GNN backbones. TRD-GNN improves over D-GNN by 1-2% and over UD-GNN by 2-5% during online evaluation. The performance of TRD-GNN is close to offline evaluation making the approach reliable. The results in red show huge drop in the performance of UD-GNN model in case of online evaluation, making it unfit for production setting.

Dataset used	**GNN-Layer Type**	**Offline Evaluation**	**Online Evaluation**
ogbn-arxiv	D-GCN	70.06	69.73
	UD-GCN	72.32	67.07
	TRD-GCN	**71.37**	**71.87**
	D-GATv2	68.72	68.89
	UD-GATv2	71.30	67.02
	TRD-GATv2	**69.54**	**69.26**
	D-GAT	69.50	68.88
	UD-GAT	72.01	66.08
	TRD-GAT	**69.74**	**69.82**
ogbn-mag	D-RGCN	36.81	32.73
	UD-RGCN	38.32	28.80
	TRD-RGCN	**37.21**	**33.83**

For the ogbn-mag heterogeneous dataset, Table 2 also reports 1.1% performance improvement from TRD-GNN. Considering the heterogeneous nature of the graph in account, TRD-GNN still outperforms baseline results. Addition-

ally, this also demonstrate the ability of TRD-GNN to adapt to different graph datasets irrespective of their nature.

We attempted to compare our performance with TGN [27] baseline leveraging the open source code available here[3]. However, the performance on ogbn-arxiv dataset was abysmal (∼7% validation accuracy), as the loss did not reduce much after 3 epochs. We tried multiple hyperparameter settings but was not able to get on-par performance and hence, chose not to include it in the main table. This is due to the fact that it requires a self-supervised pre-trained model for its dynamic node classification module. Given the high runtime of TGN as reported in Table 8, running a hyperparameter sweep was very under-productive.

On the e-commerce dataset, Table 3 showcases that TRD-GNN improves performance (in terms of AUC-PR) on top of the directed message passing graph (D-GNN) while ensuring that the time-relaxation does not cause model collapse in the online setting as compared to undirected message passing (UD-GNN). The poor performance of the undirected graph-based model during online evaluation is primarily due to the consumption of future information while training the GNN model. This is a common pitfall where GNN models can show impressive performance in offline setup due to leakages of information from the future.

Table 3: TRD-GNN as a leakage-resistant approximation of undirected graph in time-directed setting on proprietary e-commerce dataset. Results are relative and absolute numbers are not presented due to confidentiality.

GNN-layer Type	**Offline Evaluation**	**OnlineEvaluation**
D-GNN	2.48x	2.12x
UD-GNN	2.98x	1.00x
TRD-GNN	**2.62x**	**2.22x**

3.3 Analysis of TRD-GNN (EQ2)

We attempt to empirically explain the different components of TRD-GNN that help improve model performance for different GNN layers. For this, we explore the incremental homophily added by TRD sampling and the impact of the choice of combination function (γ). We perform both the analysis on the ogbn-arxiv dataset, along with a quantitative analysis on the proprietary e-commerce dataset.

Characteristics of incremental neighbors: To characterise the nature of orders where incremental edges passed on messages resulting in improved prediction, we investigate orders where the new model predicted the opposite class as compared to baseline and look at key patterns that turn up in incremental TRD neighbors on ogbn-arxiv dataset. Given that TRD-GNN acts as a deployable bridge between GNN trained on undirected graphs and those trained on

[3] https://github.com/twitter-research/tgn

directed graphs, we hypothesize incremental homophilous messages from time-relaxed neighbors to contribute to the improved performance. Multiple research studies [6,20] have shown that GNN performance is well correlated with the level of homophily in the node's neighborhood.

Table 4: % of Nodes whose performance improved due to additional same class neighbors in the backward neighborhood.

% increase in homophilous signal	**% of Nodes**
$\geq 10\%$	99.5%
$\geq 20\%$	96.8%
$\geq 30\%$	92.2%
$\geq 40\%$	87.0%
$\geq 50\%$	85.6%

Table 5: Quantitative characterisation of orders where TRD-GNN improved classification prediction over baseline

(a) False Negatives to True Positives

Type of Signal		% Nodes
Existing Signal	TRD Signal	
✓	✓	19%
✗	✓	25%
✓	✗	18%
✗	✗	36%

(b) False Positives to True Negatives

Type of Signal		% Nodes
Existing Signal	TRD Signal	
< 10	< 10	21.6%
< 10	≥ 10	32.6%
10-15	< 10	8.9%
10-15	≥ 10	30.6%
> 15	> 10	5.7%

Table 4 showcases how the nodes with additional homophilous (i.e. same class) neighbors from the backward neighborhood are the ones that help improve the model's performance. As expected, the incremental homophilous neighborhood is a key contributor in the correct classification of nodes, which was not tapped in by directed graph based sampling.

Further, for 56% of the incremental true-positives by TRD-GNN, the backward neighborhood had 100% of nodes from the same class as the node to be classified. This highlights the importance of including backward neighbor's connections during message passing.

Quantitative Evaluation on proprietary e-commerce dataset : Table 5a characterises the samples which moved from low-score False Negatives to True Positives. We consider those positive samples with baseline score ≤ 0.5 and TRD-GNN score ≥ 0.5. *Existing Signal* refers to cases where the base node was connected to same-class nodes, and *TRD Signal* refers to cases where one of $\mathcal{N}_v^b$

is also from same-class nodes. We observe that, in 44% cases, the TRD Signal added homophily signal (same class neighbor), leading to correct classification.

For the other case of False Positives to True Negatives, we define *Existing Signal* and *TRD Signal* in terms of the number of neighbors sampled for message passing by each methodology. We divide both signals into buckets and in Table 5b observe that for 63.2% orders, TRD-Signal increased homophilous neighborhood by at-least 10 neighbors.

Impact of choice of Combination Function: Table 6 highlights the consistent effect of adding backward neighbors along with forward neighbors in message passing in terms of incremental performance. We, however, observe that different choice of γ has an impact on the overall performance of the GNN. This showcases that the nature of neighbors added by TRD sampling is different from those already available. These were missed in the directed graph sampling.

Table 6: Ablation on combination function (γ) used for aggregation of forward and backward neighbors, showing the incremental benefit of backward neighbors

Forward Neighbors	Backward Neighbors	Combination Function (γ)	Accuracy (in %)
✓		-	69.73
✓	✓	SUM	70.93
✓	✓	CONCAT	70.73
✓	✓	MAX	**71.87**
✓	✓	MIN	70.57

3.4 Time Comparison (EQ3)

In this section, we compare the time taken by an epoch (including training and inference time) with D-GATv2, UD-GATv2 and TRD-GATv2 layers on ogbn-arxiv dataset and report the time in Table 7. For fair measurement, all the experiments were conducted on a 64-core machine with 488 GB RAM running Linux operating system. Table 7 draws out the fact that the average running time per epoch for TRD-GATv2 is slightly higher than that of D-GATv2 in the online setting while performing much better compared to UD-GATv2. However, during offline evaluation, TRD-GATv2 takes lesser time than both D-GATv2 and UD-GATv2. This is because TRD-GATv2 eliminates all the unused neighbors from the forward neighbors and keep only necessary neighbors in the backward neighbors set while maintaining the sanctity of timestamped nature of the graph.

Further, Table 8 brings the observation that TRD-GNN approach is much more efficient when compared to TGN. In addition, TGN requires two extra steps before finally training for the downstream node classification task, which themselves are quite expensive. Even after discounting any data preprocessing and self-supervised training on the graph, TRD-GNN is efficient, taking less than half the time required for TGN.

Table 7: Average time taken (in seconds per epoch) to run each experiment on ogbn-arxiv dataset.

Type	**Offline (UD)**	**Online (D)**
D-GATv2	874 s/epoch	128 s/epoch
UD-GATv2	2359 s/epoch	818 s/epoch
TRD-GATv2	598 s/epoch	322 s/epoch

Table 8: Run-time comparison of TRD-GATv2 with TGN on ogbn-arxiv dataset, reported in seconds per epoch.

Steps/Module	**TRD-GATv2**	**TGN**
Data-preprocessing	-	73 s
Self-Supervised Training	-	22476 s/epoch
Node classification Training	322 s/epoch	684 s/epoch

4 Related Work

GNNs have become ubiquitous to graph based modeling which is a very common use-case in industry. The increased penetration of e-commerce and social networks in human life has increased the scale at which GNNs will be employed. GNNs today power different business critical solutions that involve user/item recommendations [26,40], feed ranking [11], fraud detection [15,29] etc. As discussed in previous sections and demonstrated by experiments section, time-stamp acknowledging training of GNNs is critical for reliable deployable GNN solution. We deep dive into the literature on directed and timestamped graphs beyond TGNs as discussion on them is presented in Sec. 2.4.

In [4,5], the authors work on a strongly connected graph by constructing a directed laplacian using random walks to leverage the GCN for directed graphs. [31] used the PageRank based constructed laplacian in [4, 5] in place of the random walk based one. These methods are not popular among the industry due to scale challenges involved in deployment. TGNs [12, 23, 27] are based on the dynamism that allows node-wise events where new nodes are added or removed as time progresses and edge-incidents where edges between nodes are either added or removed from the graph. One key aspect of TGN was the use of RNN-based memory component that updated the representation of nodes as newer information comes in with every passing time. While the approach may seem generic, it is hard to fine-tune and take considerably longer than static graph methods. [41] proposed time-aware GNN for aligning entities between the temporally evolving knowledge graphs. They embed entities, relations and timestamps of different KGs into a single space and use GNNs to learn entity representations in the same space. They present a time-aware attention mechanism that assigns different weights to different nodes with orthogonal transformation matrices computed from embeddings of the relevant relations and timestamps in a neighborhood.

Spatio-temporal graph learning [22,42] is an efficient structure to characterize the relations between different nodes in a specified spatial and temporal range. They assume a fixed number of nodes while training and testing and expect changes only in the adjacency of the graph. These assumptions of fixed number of nodes make it difficult to extend such approaches to practical industry setting where nodes are continuously added to the system as new users join the service.

From a production deployment perspective, there is no work, to the best of our knowledge, that discusses directionality in timestamped graph and creates any solution around it. A large majority of GNN solutions proposed for fraud/abuse detection [30, 37, 39] do not incorporate the timestamped aspect of observation. Some, however, work on directed graphs [34,38] where the direction is an outcome of the underlying data generation process. Our work is the first that discusses the breakdown of undirected graphs in online production setting and proposes a simple yet effective bridge between timestamp based directed and undirected graphs, resulting in improved performance. We also create benchmark performances on public datasets that can be leveraged by community to further improve along this direction.

5 Application to Industry

Industry specific online ML models are time critical in nature. Therefore, it is necessary to make sure that online performance matches the expected offline performance. GNNs are particularly susceptible to label-leakage in industry applications. TRD-GNN provides a simple yet effective mechanism to make the graph partially undirected and eliminate label leakage with consistency in training and evaluation. We showcase that, with TRD-GNN (Algorithm 1), offline performance of model is in-line with online performance making the GNN model training process reliable, thus delivering the business objective of abuse detection (Table 3). We deployed the TRD-GNN in an offline manner where the graph is updated with new nodes and edges at a fixed cadence, and the GNN model is used for identifying abuse.

6 Conclusion

We propose an effective relaxation technique (TRD-GNN) for timestamped directed graphs that is resilient to label-leakage and helps improve classification performance, making the model reliable to use in production. The idea of TRD-GNN is agnostic to GNN layer and type of graph and is extensible to all GNN-based tasks where time direction is critical, which is typical to industry production use-cases. The relaxation lies on a spectrum with directed sampling on one end and undirected on the other. We also present an analysis which answers why TRD-GNN is able to improve over vanilla-directed and undirected GNNs.

References

1. Biewald, L.: Experiment tracking with weights and biases, 2020. Software available from wandb. com **2**(5) (2020)
2. Breuer, A., Eilat, R., Weinsberg, U.: Friend or faux: Graph-based early detection of fake accounts on social networks. In: Proceedings of The Web Conference 2020. p. 1287–1297. WWW '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3366423.3380204, https://doi.org/10.1145/3366423.3380204
3. Brody, S., Alon, U., Yahav, E.: How attentive are graph attention networks? In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=F72ximsx7C1
4. Cao, D., Li, J., Ma, H., Tomizuka, M.: Spectral temporal graph neural network for trajectory prediction. In: 2021 IEEE International Conference on Robotics and Automation (ICRA). pp. 1839–1845. IEEE (2021)
5. Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., et al.: Spectral temporal graph neural network for multivariate time-series forecasting. Advances in neural information processing systems **33**, 17766–17778 (2020)
6. Chanpuriya, S., Musco, C.: Simplified graph convolution with heterophily (2022)
7. Chen, M., Wei, Z., Ding, B., Li, Y., Yuan, Y., Du, X., Wen, J.: Scalable graph neural networks via bidirectional propagation. CoRR **abs/2010.15421** (2020), https://arxiv.org/abs/2010.15421
8. Chien, E., Chang, W.C., Hsieh, C.J., Yu, H.F., Zhang, J., Milenkovic, O., Dhillon, I.S.: Node feature extraction by self-supervised multi-scale neighborhood prediction. In: International Conference on Learning Representations (ICLR) (2022)
9. Cui, L., Tang, X., Katariya, S., Rao, N., Agrawal, P., Subbian, K., Lee, D.: Allie: Active learning on large-scale imbalanced graphs. In: The Web Conference 2022 (2022), https://www.amazon.science/publications/allie-active-learning-on-large-scale-imbalanced-graphs
10. Dhawan, S., Gangireddy, S.C.R., Kumar, S., Chakraborty, T.: Spotting collective behaviour of online frauds in customer reviews. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. p. 245–251. IJCAI'19, AAAI Press (2019)
11. Fan, W., Ma, Y., Li, Q., Wang, J., Cai, G., Tang, J., Yin, D.: A graph neural network framework for social recommendations. IEEE Transactions on Knowledge and Data Engineering **34**(5), 2033–2047 (2020)
12. Fan, Y., Ju, M., Zhang, C., Ye, Y.: Heterogeneous temporal graph neural network. In: Proceedings of the 2022 SIAM International Conference on Data Mining (SDM). pp. 657–665. SIAM (2022)
13. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs (2017). https://doi.org/10.48550/ARXIV.1706.02216, https://arxiv.org/abs/1706.02216
14. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open graph benchmark: Datasets for machine learning on graphs. arXiv preprint arXiv:2005.00687 (2020)
15. Huang, M., Liu, Y., Ao, X., Li, K., Chi, J., Feng, J., Yang, H., He, Q.: Auc-oriented graph neural network for fraud detection. In: Proceedings of the ACM Web Conference 2022. p. 1311–1321. WWW '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3485447.3512178, https://doi.org/10.1145/3485447.3512178

16. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
17. Kumar, I., Hu, Y., Zhang, Y.: Eflec: Efficient feature-leakage correction in gnn based recommendation systems. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 1885–1889. SIGIR '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3477495.3531770, https://doi.org/10.1145/3477495.3531770
18. Li, A., Qin, Z., Liu, R., Yang, Y., Li, D.: Spam review detection with graph convolutional networks. pp. 2703–2711 (11 2019). https://doi.org/10.1145/3357384.3357820
19. Li, G., Müller, M., Ghanem, B., Koltun, V.: Training graph neural networks with 1000 layers. In: International Conference on Machine Learning (ICML) (2021)
20. Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., Qian, W.: Finding global homophily in graph neural networks when meeting heterophily (2022)
21. Liao, P., Zhao, H., Xu, K., Jaakkola, T., Gordon, G.J., Jegelka, S., Salakhutdinov, R.: Information obfuscation of graph neural networks. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 139, pp. 6600–6610. PMLR (18–24 Jul 2021), http://proceedings.mlr.press/v139/liao21a.html
22. Liu, X., Liang, Y., Zheng, Y., Hooi, B., Zimmermann, R.: Spatio-temporal graph contrastive learning. arXiv preprint arXiv:2108.11873 (2021)
23. Min, S., Gao, Z., Peng, J., Wang, L., Qin, K., Fang, B.: Stgsn—a spatial–temporal graph neural network framework for time-evolving social networks. Knowledge-Based Systems **214**, 106746 (2021)
24. Olatunji, I., Nejdl, W., Khosla, M.: Membership inference attack on graph neural networks. pp. 11–20 (12 2021). https://doi.org/10.1109/TPSISA52974.2021.00002
25. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
26. Qiu, R., Li, J., Huang, Z., Yin, H.: Rethinking the item order in session-based recommendation with graph neural networks. In: Proceedings of the 28th ACM international conference on information and knowledge management. pp. 579–588 (2019)
27. Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., Bronstein, M.: Temporal graph networks for deep learning on dynamic graphs (2020). https://doi.org/10.48550/ARXIV.2006.10637, https://arxiv.org/abs/2006.10637
28. Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks (2017)
29. Shi, F., Cao, Y., Shang, Y., Zhou, Y., Zhou, C., Wu, J.: H2-fdetector: A gnn-based fraud detector with homophilic and heterophilic connections. In: Proceedings of the ACM Web Conference 2022. p. 1486–1494. WWW '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3485447.3512195, https://doi.org/10.1145/3485447.3512195
30. Tang, J., Li, J., Gao, Z., Li, J.: Rethinking graph neural networks for anomaly detection. In: International Conference on Machine Learning (2022)

31. Tong, Z., Liang, Y., Sun, C., Li, X., Rosenblum, D., Lim, A.: Digraph inception convolutional networks. Advances in neural information processing systems **33**, 17907–17918 (2020)
32. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks (2018)
33. Wang, B., Gong, N.Z., Fu, H.: Gang: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs. In: 2017 IEEE International Conference on Data Mining (ICDM). pp. 465–474 (2017). https://doi.org/10.1109/ICDM.2017.56
34. Wang, B., Gong, N.Z., Fu, H.: Gang: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs. In: 2017 IEEE International Conference on Data Mining (ICDM). pp. 465–474 (2017). https://doi.org/10.1109/ICDM.2017.56
35. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
36. Wang, Y., Jin, J., Zhang, W., Yu, Y., Zhang, Z., Wipf, D.: Bag of tricks for node classification with graph neural networks. arXiv preprint arXiv:2103.13355 (2021)
37. Wang, Y., Zhang, J., Huang, Z., Li, W., Feng, S., Ma, Z., Sun, Y., Yu, D., Dong, F., Jin, J., et al.: Label information enhanced fraud detection against low homophily in graphs. arXiv preprint arXiv:2302.10407 (2023)
38. Wu, J., Yao, M., Wu, D., Chi, M., Wang, B., Wu, R., Fu, X., Meng, C., Wang, W.: Dedgat: Dual embedding of directed graph attention networks for detecting financial risk (03 2023). https://doi.org/10.48550/arXiv.2303.03933
39. Wu, Q., Chen, Y., Yang, C., Yan, J.: Energy-based out-of-distribution detection for graph neural networks. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=zoz7Ze4STUL
40. Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T.: Session-based recommendation with graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence. vol. 33, pp. 346–353 (2019)
41. Xu, C., Su, F., Lehmann, J.: Time-aware graph neural network for entity alignment between temporal knowledge graphs. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 8999–9010. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (Nov 2021). https://doi.org/10.18653/v1/2021.emnlp-main.709, https://aclanthology.org/2021.emnlp-main.709
42. Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv preprint arXiv:1709.04875 (2017)

Ethical Statement

Our work is a generic framework to boost the performance of timestamped directed graph neural network. This work applies to GNNs generally and any graph-structured data specifically. Its ethical impacts (including positive and negative) depend on the specific domain of the data. We conduct our experiments on publicly available datasets and real-world datasets in compliance with fair-use clauses.