Appendices for: GTEA: Inductive Representation Learning on Temporal Interaction Graphs via Temporal Edge Aggregation

Siyue Xie[1⋆], Yiming Li[1⋆], Da Sun Handason Tam[1], Xiaxin Liu[2], Qiufang Ying[2], Wing Cheong Lau[1], Dah Ming Chiu[1], and Shouzhi Chen[2]

[1] The Chinese University of Hong Kong
[2] Tencent Technology Co.Ltd

1 More Discussions and Comparisons on Related Works

1.1 Comparisons on Temporal Inductive Models

In real-world environments, the topology of a TIG usually changes as time goes. For example, in an online social network, not only existing users interact with each other, but also new users be added and they will communicate with others. Such situations require that models adaptive to real-world product scenarios should be with an inductive nature and capable of dealing with temporal dynamics.

Many previous methods attempt to learn from temporal graphs but most of them [12, 24] can only work for transductive tasks. But there still some recent works attempt to inductively learn representations for temporal graphs. TDGNN [14] inherits the working pipeline of GraphSAGE [3] and extend it to temporal graphs by incorporating the time information of each interaction. The object of each interaction of a target node is treated as an independent neighbor. Neighbors aggregation can therefore be applied to all interactions, accompanied with an time-exponential decreasing weight to attenuate the importance of a past event. The framework of TGAT [23] is similar to TDGNN but instead introduce a self-attention mechanism to weight different interactions/ neighbors for aggregation. In addition, a time-encoding method is incorporated to embed time information. CAWs [22] proposes to utilize causal anonymous walks on temporal graphs to generate node embeddings, which helps mine some local motifs of the graph structure.

However, there is still a common drawback of the aforementioned inductive methods: the identity information of interactions is ignored or even purposefully removed. However, in TIGs, we argue that the identity of each interaction object can help models learn more discriminative node representations, as the illustrative example we shown in Figure 1 in our main text. In CAW [22], to capture temporal motifs structures, different nodes with the same causal walk orders will

⋆ Equal contributions.

result in similar representations. TDGNN [14] and TGAT [23] lumps all interactions of a target node for aggregation. Although the self-attention mechanism can help TGAT to highlight some more important interactions, it is unable to distinguish different interactions triggered by the same neighbor. In contrast, our proposed model, GTEA, learns an embedding to represent a sequence of interactions between a pair of adjacent nodes. Interactional patterns encoded in an edge embedding are consistently extracted from the same instance, which preserves the discriminative identity information for aggregation. Experimental results shown in Section 4 in main text also demonstrate the superiority of GTEA over TGAT.

1.2 Comparisons on Dynamic Embedding Methods

Dynamic embedding methods are commonly used to learn node representations for TIGs. Specifically, a node embedding is usually randomly-initialized and the model will hold a memory to store the embeddings (or status related to embeddings) of all nodes. By observing the interactions happened along time, node embeddings can be dynamically updated.

TigeCMN [24] leverages the recent advances in memory networks to learn temporal interactional features as well as node multi-faceted properties. MNCI [9] additionally learns a group of community embeddings to model the community influence on node embeddings. JODIE [7] focuses on user-item bi-partite graphs and utilizes two independent RNN models to learn node representations by tracking the embedding trajectories. TGN [16] additionally introduces a time-encoder and implements a training strategy that allows it to update and learn node embeddings with efficient parallelization.

Although dynamic embedding methods have achieved great success, the drawbacks can also be obvious. First, since a memory is required to maintain the embedding of existing nodes, some of the methods are hard to generalize or even fail to work when there is new nodes added (i.e., inductive tasks). Second, the corresponding node embeddings are always updated once an interaction is observed. The status of a node may change frequently, which is not expected for some downstream tasks, e.g., anomaly detection or phishing node detection, where only few interactions of the entire interaction history carry critical hints. Such a updating paradigm may fail to capture the most discriminative patterns in the interactions. Consequently, it is sometimes hard to yield a general representations that can depict the profile of a user, which is very important in some tasks, e.g., to identify the role of a user in a TIG as in our experiments. Last but not least, the model have to maintain a big and heavy memory to remember and update all nodes it observed. The condition may get worse or even inapplicable when it comes to a large-scale graph with million or billion nodes. The above drawbacks are also the reasons why these methods are inapplicable to the dataset we mainly concerned, i.e., the Mobile-Pay dataset.

Compared with the aforementioned models, GTEA is designed to work for inductive tasks (by learning model weights instead of learning node embeddings

directly), which easily handles the cases when the graph topology evolves. Moreover, GTEA models the interactional dynamics by observing the entire interaction history between a pair of nodes. This helps it mine temporal relational patterns from a global view and more helpful to summarize the status of each node. Results on different tasks in our experiments also demonstrate the effectiveness of GTEA.

2 Technical Details of the Proposed Method

In this section, we provide more technical details of the key modules of GTEA.

2.1 Interaction Dynamics Modeling with Different Sequence Models

We implement different GTEA variants with two sequence models, i.e., LSTM and Transformer, for interactional dynamics modeling. In the following, we specify the details.

Temporal Dynamics Modeling with LSTM LSTM [4] is implemented to capture the patterns of consecutive interactions between a pair of nodes. Concretely, let's denote $[\mathbf{e}_{uv}^1, ..., \mathbf{e}_{uv}^{S_{uv}}]$ as the interaction sequence of edge (u, v), where S_{uv} indicates the total number of events between a specific node pair. Note that S_{uv} can be different for different node pairs. We therefore adapt the LSTM model to encode the interaction sequence of edge (u, v) as follows (we drop the subscript $_{uv}$ for notational convenience):

$$\mathbf{i}[k] = \sigma\left(\mathbf{W}_i\mathbf{e}^k + \mathbf{U}_i\mathbf{h}[k-1] + \mathbf{b}_i\right) \tag{1}$$

$$\mathbf{f}[k] = \sigma\left(\mathbf{W}_f\mathbf{e}^k + \mathbf{U}_f\mathbf{h}[k-1] + \mathbf{b}_f\right) \tag{2}$$

$$\overline{\mathbf{c}}[k] = \tanh\left(\mathbf{W}_c\mathbf{e}^k + \mathbf{U}_c\mathbf{h}[k-1] + \mathbf{b}_c\right) \tag{3}$$

$$\mathbf{c}[k] = \mathbf{f}[k] \odot \mathbf{c}[k-1] + \mathbf{i}[k] \odot \overline{\mathbf{c}}[k] \tag{4}$$

$$\mathbf{o}[k] = \sigma\left(\mathbf{W}_o\mathbf{e}^k + \mathbf{U}_o\mathbf{h}[k-1] + \mathbf{b}_o\right) \tag{5}$$

$$\mathbf{h}[k] = \mathbf{o}[k] \odot \tanh\left(\mathbf{c}[k]\right) \tag{6}$$

$\mathbf{e}^k$ represents the k-th step's input of the LSTM. In GTEA, $\mathbf{e}^k$ corresponds to the features of the k-th interaction event. $\mathbf{i}[k]$, $\mathbf{f}[k]$, and $\mathbf{o}[k]$ are state vectors of the input gates, forget gates and output gates respectively. $\mathbf{c}[k]$ is the memory cell and $\mathbf{h}[k]$ indicates the hidden state. $\sigma(\cdot)$ and $\tanh(\cdot)$ represent the sigmoid and hyperbolic tangent activation functions respectively. $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_c, \mathbf{W}_o, \mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_c, \mathbf{U}_o, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_c, \mathbf{b}_o$ are trainable parameters of the network. We take the last hidden output $\mathbf{h}_{uv}[S]$ of the LSTM as the edge embeddings $\tilde{\mathbf{e}}$. Note that $\mathbf{e}^k$ will be replaced by $\hat{\mathbf{e}}^k$ when the edge features are enhanced by the time-embedding.

Temporal Dynamics Modeling with Transformer Transformer [19] is another popular architecture for sequence modeling. The key component of the Transformer architecture is the self-attention mechanism, where a self-attention function maps a query and a set of key-value pairs to an output. In particular, given a pair of nodes (u, v), we formulate its interaction sequence as a matrix: $\mathbf{E}_{uv} = [\mathbf{e}^1, ..., \mathbf{e}^{S_{uv}}]^\intercal$. Then the projection of the interaction sequence on the query, key and value space can be computed as:

$$\mathbf{Q} = \mathbf{E}_{uv}\mathbf{W}_Q, \tag{7}$$

$$\mathbf{K} = \mathbf{E}_{uv}\mathbf{W}_K, \tag{8}$$

$$\mathbf{V} = \mathbf{E}_{uv}\mathbf{W}_V. \tag{9}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ denotes the queries, keys and values matrices, respectively. $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ are learnable parameters. Then the "Scaled Dot-Product Attention" function can be defined as:

$$\tilde{\mathbf{E}}_{uv} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\mathbf{T}}{\sqrt{D_E}})\mathbf{V} \tag{10}$$

where D_E is the hidden dimensions of a query vector. Since Transformer implicitly computes the mutual correlations of all interactions and the last interaction empirically reflects the latest relation status of a node pair, it is sufficient to represent the edge embedding $\tilde{\mathbf{e}}_{uv}$ by the S-th row of the $\tilde{\mathbf{E}}_{uv}$, i.e., the embedding corresponding to the last interaction. Note that for time-encoder-enhanced variants, the input of Transformer, i.e., the interaction matrix $\mathbf{E}_{uv}$, will also be replaced by $\hat{\mathbf{E}}_{uv} = [\hat{\mathbf{e}}^1, ..., \hat{\mathbf{e}}^{S_{uv}}]^\intercal$.

In our experiments, we implement a multi-head attention transformer for GTEA. The final edge embedding is yielded by the mean of each attention head.

2.2 Additional Analyses on the Time-embedding

In the interaction sequence of an edge (u, v), the time gap of any two consecutive interactions can vary with great differences. In order to capture more fine-grained time-related patterns of an interaction sequence, e.g., time periodicity, we follow the ideas of Time2Vec [11] (**T2V**) to generate a time-embedding to further enhance the edge interaction features. The key idea of the time-embedding is to map the timestamp of each interaction from a scalar to a vector, which is capable of encoding different time-related properties.

Specifically, we generate a time embedding with l dimensions for any given timestamp t as follows:

$$\tau(t)[i] = \begin{cases} \omega_i t + \varphi_i, & \text{if } \ i = 0. \\ \cos\left(\omega_i t + \varphi_i\right), & \text{if } \ 1 \leq i \leq l. \end{cases} \tag{11}$$

where $\omega'_0, ...\omega'_l$ and $\varphi'_0, ..., \varphi'_l$ are trainable parameters.

Algorithm 1 Sparsemax Transformation

1: **Input:** $\boldsymbol{z}$
2: Sort $\boldsymbol{z}$ as $z_{(1)} \geq ... \geq z_{(K)}$
3: Find $k(\mathbf{z}) := \max\{k \in [K] | 1 + kz_{(k)} \geq \sum_{j \leq k} z_{(j)}\}$
4: Define $\gamma(\mathbf{z}) = \frac{\sum_{j \leq k} z_{(j)} - 1}{k(\mathbf{z})}$
5: **Output:** $\boldsymbol{p}$ s.t. $p_i = [z_i - \gamma(\boldsymbol{z})]_+$

On the theoretical side, $\tau(t)$ can be viewed as an extension of Random Fourier Features (RFF) [15]. If we define:

$$\gamma(t) = \sqrt{\frac{2}{k}}[\cos(\omega_1' t + \varphi_1'), ..., \cos(\omega_k' t + \varphi_k')]^\intercal \tag{12}$$

where $\omega_1', ...\omega_k'$ are i.i.d samples from some probability distribution $p(\omega)$ and $\varphi_1', ..., \varphi_k'$ are i.i.d samples from the uniform distribution on $[0, 2\pi]$. Then, as a consequence of the Bochner's theorem from harmonic analysis, it can be shown that $\mathbb{E}[\gamma(t_1)^\intercal \gamma(t_2)] = \phi(t_1, t_2)$ for some positive definite shift-invariant kernel $\phi(t_1, t_2) = \phi(t_1 - t_2)$. Thus, the time-embedding can be regarded as RFF with tunable phase-shifts and frequencies.

Empirically, the i-th element (except when $i = 0$) of $\tau(t)$ is defined by a harmonic function, whose value recurs with a period of $\frac{2\pi}{\omega_i}$. Therefore, $\tau(t)$ has the capability to encode the periodicity of a sequence of timestamps. For example, if t is measured by 'hours' and some interactions between two specific nodes always happen at the same time of a day, then the time-embedding can encode the daily recurrence by making $\omega_i = \frac{2\pi}{24} = \frac{\pi}{12}$. In other words, the interactions occur at a specific time will share the same embedding value. In our experiment, we let GTEA to learn such time-related patterns in an end-to-end manner based on different downstream tasks.

It is worth noting that sinusoidal activation is also found to be suitable for representing complex natural signals [18] with high precision. In addition, sinusoidal functions with fixed frequencies and phase-shifts have also been used in the Transformer model [19] as positional encodings. However, it has been shown that learning the frequencies and phase-shifts as in **T2V** rather than fixing them can achieve better performance.

2.3 Details of the Sparsity-Inducing Attention

In GTEA, we propose to learn sparse attentive weights for neighborhood aggregation. The process can be formulated as follows:

$$\tilde{\alpha}_{uv} = \mathbf{a}^\intercal \mathbf{h}_{uv}, \quad \mathbf{h}_{uv} = Enc_a([\hat{\mathbf{e}}_{uv}^1, ..., \hat{\mathbf{e}}_{uv}^{S_{uv}}]), \tag{13}$$

$$\boldsymbol{\alpha}_{u:} = \text{Sparse}(\tilde{\boldsymbol{\alpha}}_{u:}), \tag{14}$$

In particular, we learn an attention embedding $\mathbf{h}_{uv}$ for each interaction sequence between the target node and a neighboring node. We then map the attention

embedding into a scalar by computing the inner-product of it with a learnable vector $\mathbf{a}$, which yields an attention measurement for each edge. We finally apply a sparsification operator, Sparse$(\cdot)$ to sparsify the attentive weights, which essentially discard some noisy neighbors for the neighborhood aggregation. In GTEA, we adopt Sparsemax [10] as our sparsification operator. Algorithm 1 shows the detailed sparsification steps. Note that the sparsification operator Sparse$(\cdot)$ is not unique and can be replaced by any gradient-continuous operators if suitable.

2.4 Additional Analyses on Temporal Edge Aggregation

To obtain an expressive node representation, we integrate edge embeddings with neighbor node embeddings. To jointly learn node and edge information, [23, 24] simply concatenates the node embedding with the edge embedding.

However, **we argue that: the operation of simply concatenating node and edge embeddings is inferior for jointly learning instance-aware features for neighborhood aggregations.** We empirically show it as as follows: as shown in Equation (15), when concatenating node and edge embeddings, we can reformulate the aggregation function by splitting the transformation weight matrix W into two independent matrices $W = [W_1, W_2]$:

$$\begin{aligned} \mathbf{z}_{\mathcal{N}(u)}^{(l)} &= \sum_{v \in \mathcal{N}(u)} \mathbf{W}([\mathbf{z}_v^{(l-1)} || \tilde{\mathbf{e}}_{uv}]]) \\ &= \mathbf{W}_1 \sum_{v \in \mathcal{N}(u)} \mathbf{z}_v^{(l-1)} + \mathbf{W}_2 \sum_{v \in \mathcal{N}(u)} \tilde{\mathbf{e}}_{uv} \end{aligned} \tag{15}$$

Equation (15) shows that: the neighboring node embeddings and the corresponding edge embeddings are essentially summed independently [8]. Node embeddings are fully isolated with their corresponding edge embeddings, which makes the model agnostic to the identity (node) of an edge embedding and therefore messes up the interactional information from all neighbors. In other words, it is not a good choice to jointly learn identity-aware features by applying naive concatenation of node and edge embeddings.

To get rid of such a drawback, we propose to introduce additional projection functions, e.g., a MLP, before aggregation to address the problem:

$$\mathbf{z}_{\mathcal{N}(u)}^{(l)} = \sum_{v \in \mathcal{N}(u)} \alpha_{uv} \mathrm{MLP}_1([\mathbf{z}_v^{(l-1)} || \tilde{\mathbf{e}}_{uv}]) \tag{16}$$

$$\mathbf{z}_u^{(l)} = \mathrm{MLP}_2([\mathbf{z}_u^{(l-1)} || \mathbf{z}_{\mathcal{N}(u)}^{(l)}]) \tag{17}$$

where MLP_1 and MLP_2 are two independent MLP models. Equation (16) applies attention weighted sum to aggregate both the node embeddings $\mathbf{z}_u^{(l-1)}$ and the associated edges embedding vector $\tilde{\mathbf{e}}_{uv}$ of its neighbors at layer $(l-1)$, where α_{uv} is derived from the sparse attention mechanism. We can show that:

Theorem: The representation ability of the aggregation operator defined in Equation (16) is strictly more powerful than that of Equation (15).

Proof: Let $\alpha_{uv} = \frac{1}{|\mathcal{N}(u)|}$ and degrade MLP_1 to a single-layer fully-connected neural networks without non-linear activations, which can be denoted as $\text{MLP}_1(\mathbf{x}) = |\mathcal{N}(u)|\mathbf{W}\mathbf{x}$. Then Equation (16) is degraded to the same form as Equation (15). Therefore, the representation ability of Equation (16) is naturally more powerful than that of Equation (15).

MLP_1 projects the concatenated embedding (of node and edge from the same instance/ neighbor) to a hidden representation space, which forces the model to correlate the node and its corresponding edge information before aggregation. Such an operator empirically preserves the identity information for TIGs.

After computing the neighboring feature vector $\mathbf{z}_{\mathcal{N}(u)}^{(l)}$, we concatenate it with the target node's embedding in the previous layer $\mathbf{z}_u^{(l-1)}$ and feed it through another MLP, i.e., MLP_2, as in Equation (17). By combining the aggregation module and the pairwise temporal sequence model, We can recursively stack the framework at a depth of L layers to learn node representations with high-level semantics.

3 Supplementary Materials of Experiments

In our experiments, we evaluate our proposed model by the node classification task on a real-world industrial dataset (denoted as Mobile-Pay), provided by a major mobile payment provider. The classification problem is formulated as anomaly detection, i.e., a binary classification task. We additionally apply GTEA on two other real-world datasets to evaluate its performance for node classifications. Furthermore, to evaluate the generalization ability of representation performance, we extend GTEA to the task of future link predictions. ***Note that we align the experimental setups of other datasets with Mobile-Pay in order to follow the real-world product scenarios. Therefore, some previous methods [7, 12, 16, 24] that cannot fit such settings are not implemented for comparison. Since our experimental settings are different from some other works [7, 16], their reported results on the same dataset are unsuitable to directly compare with our results.***

3.1 Datasets

We conduct node classifications on three datasets: Mobile-Pay, Phish-S and Phish-L. For future link predictions, we evaluate GTEA on the Wikipedia and Reddit datasets. In the following, we introduce each of these datasets and elaborate the experimental settings. A brief statistics of these five datasets are summarized in Table 1.

Table 1. Dataset Summary

Dataset	# Nodes	# Edges	# Interactions	# Labeled Nodes (# Samples)
Mobile-Pay	2,143,844	4,568,936	21,326,122	6,688
Phish-S	1,329,729	2,161,573	6,794,521	3,360
Phish-L	2,973,489	5,355,155	184,398,820	6,165
Wikipedia	9,227	20,922	157,474	26,678
Reddit	10,984	103,956	672,447	156,314

Mobile Payment Network Dataset With the rapid growth of mobile payment services, illicit users bring critical challenges to both service provides and regulators. We therefore collect a dataset, named as *Mobile-Pay*, from a major mobile payment provider, with 3347 users labeled as "illicit" and 3341 normal users, to form a TIG for anomaly detection. In the graph, a node indicates a user. Transactions between two users are represented as an edge, which corresponds to a sequence of interaction events. Each node is associated with some static attributes to describe the user, while each interaction of an edge records some basic information of a transaction, e.g., amount and time. After filtering out edges with few transaction counts and nodes with extremely large degrees, the resultant graph has 2,143,844 nodes and 21,326,122 edges. All user-related information is anonymized for privacy security concerns. Node and edge features, such as transaction amount, are normalized before experiments. **The goal of GTEA on this dataset is to identify the role of a user (whether it is an illicit user or normal user), given the history of all transaction interactions in the past.**

Phishing Dataset We collect two different Ethereum Phishing Datasets provided by [1], referred as *Phish-S* ('S' stands for 'small') and *Phish-L* ('L' stands for 'large') respectively[3]. **The goal on these two datasets is to determine whether an account in the Ethereum network is a phishing account.** To build the graph, each account is viewed as a node and transaction sequences between two nodes indicate an edge. For each user account, we compute some statistical features as node attributes based on the transaction records it involved. In *Phish-S*, we obtain 1,660 phishing nodes and 1,700 normal nodes. In *Phish-L*, 1,165 nodes are annotated as phishing users and we randomly select 5,000 nodes from the rest as normal users, which sums up to 6,665 labeled nodes.

Wikipedia Dataset Wikipedia is the world's most widely used online encyclopedia. [7] collects the editing events in Wikipedia from the internet[4] and form the *Wikipedia* dataset. Each event (interaction) records the editing content and timestamp of a user on a wiki-entry. The text editing content is vectorized as

[3] Raw data: https://www.kaggle.com/xblock/ethereum-phishing-transaction-network
[4] http://snap.stanford.edu/jodie/wikipedia.csv

an embedding in advance, which is used as the raw features of each interaction in an edge. As for nodes, we use a randomly initialized vector to represent node attributes. The dataset contains 8,277 users and 1,000 items (wiki-entry). There are totally 157,474 interaction between users and items. We sort the interactions based on their timestamps and take the first 50% of interactions to construct the graph, where users or items are nodes and an interaction sequence of a user-item pair induces an edge. **The goal of GTEA on this dataset is to determine whether a user will interact with an item at *any time of the future*, given all the interaction history in the constructed graph.** Therefore, we randomly split the remaining interactions into a training set (60%), validation set (20%) and test set (20%). Node pairs found interacted in the train/ validation/ test set are regarded as positive samples. To construct negative samples, we randomly pick one item that does not have interactions with the user (of each positive sample) to form a negative user-item pair. The resultant graph has 9,277 nodes and 20,922 edges with 26,678 (positive and negative) samples in total.

Reddit Dataset The Reddit dataset is also provided by [7], which contains 1,000 users and 984 items with 672,447 interactions among them[5]. **The setting on Reddit is similar to that of Wikipedia.** The resultant graph has 10,984 nodes and 103,956 edges with 156,314 samples.

On each of the datasets, we apply five-fold cross-validation strategy to evaluate our model.

3.2 Implementation Details of Compared Methods and Different GTEA Variants

To evaluate the performance of GTEA, we compare our model with state-of-the-art methods that are able to handle large-scale graphs. Specifically, there are six baseline models and five GTEA variants, which can be categorized as follows:

- **GNNs learned only with node features**. These methods are baseline GNNs that learn node embeddings without edge features or temporal information. We take GCN [6], GraphSAGE [3], GAT [20] as the representative models for comparison. Since these methods cannot utilize time-related information of a TIG, i.e., interaction sequences are not incorporated during training, the learned node embeddings only reflect the topological dependencies.
- **Methods incorporate edge features**. This category of models incorporate node features as well as statistical edge features that are manually derived from temporal interactions between each node pair. We implement ECConv [17] and EGNN [2] as the representatives for comparison. We also

[5] http://snap.stanford.edu/jodie/reddit.csv

Algorithm 2 GTEA Framework (One Forward-Propagation Step)

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; node features $x_u, \forall u \in \mathcal{V}$; temporal edge sequences $\mathbf{e}_{uv}, \forall (u, v) \in \mathcal{E}$; depth of GNN backbone L; aggregation functions MLP_1 and MLP_2; neighborhoods of node u: $\mathcal{N}(u)$; edge embedding model Enc_i and attention model Enc_a; trainable attention weight vector $\mathbf{a}$.
Output: Node embedding $\mathbf{z}_u^{(L)}$

$\mathbf{z}_u^{(0)} \leftarrow x_u$;
for $l = 1, 2, ..., L$ **do**
  $\tilde{\mathbf{e}}_{uv} \leftarrow Enc_i(e_{uv}), \forall v \in \mathcal{N}(u)$;
  $\mathbf{h}_{uv} \leftarrow Enc_a(e_{uv}), \forall v \in \mathcal{N}(u)$;
  $\tilde{\alpha}_{uv} \leftarrow \mathbf{a}^\intercal \mathbf{h}_{uv}$
  $\alpha_{uv} \leftarrow \text{Sparse}_v(\tilde{\alpha}_{u:})$
  $\mathbf{z}_{\mathcal{N}(u)}^{(l)} \leftarrow \sum_{v \in \mathcal{N}(u)} \alpha_{uv} \text{MLP}_1([\mathbf{z}_v^{(l-1)} || \tilde{\mathbf{e}}_{uv}])$;
  $\mathbf{z}_u^{(l)} \leftarrow \text{MLP}_2([\mathbf{z}_u^{(l-1)} || \mathbf{z}_{\mathcal{N}(u)}^{(l)}])$
end for
return $\mathbf{z}_u^{(L)}$;

implement a variant of GTEA named GTEA_{HE}, which replaces the automatically learned edge-embedding with handcrafted interaction-statistics-based edge features. The handcrafted edge features include the mean, minimum, maximum and standard deviation amount of feature values of each interaction sequence.

- **GNNs with temporal learning**. We also compare GTEA with TGAT [23], a state-of-the-art representation learning scheme for temporal graphs learning.
- **GTEA variants**. We implement different GTEA variants regarding the sequence model. Specifically, the variants equipped with LSTM and Transformer are denoted by GTEA_{L} and GTEA_{TX} respectively. Besides, the variants enhanced by the time encoder are marked with a '+T' suffix, including GTEA_{L}+T and GTEA_{TX}+T.
  Note that we do not implement dynamic embedding methods for comparisons. Reasons are elaborated in Appendix 1.2.

3.3 Training Details

We summarize the algorithm of GTEA in Algorithm 2. All the algorithms are implemented using Pytorch [13] and DGL v0.5 [21]. For fair comparison, we tuned all baselines and GTEA variants with different hyperparameter settings. We search learning rate in {0.0001, 0.001, 0.01}, number of GNN layers in {1, 2, 3}, hidden embedding dimension in {32, 64, 128} for both nodes and edges. For models using LSTM, i.e. our GTEA_{L} and GTEA_{L}+T variants, the number of temporal layers is searched from {1, 2, 3}. For Transformer-based variants, i.e. GTEA_{TX} and GTEA_{TX}+T, the number of transformer layers is searched among {1, 2, 3}, and the number of attention heads is set to 4. To make it train on large-scale graphs, we apply mini-batch training with the neighbor sampling

strategy [3] for all models if applicable. We tune the batch size with values in {32, 64, 128}, and the neighbors sampling size in {5, 10, 25}. Adam [5] is used as the optimizer in our training stage. For all other models implemented for comparisons, their tuning process is the same as that of GTEA. We tune the hyperparameters based on the validation set and we report the best accuracy and macro F1 score on the test set.

Our experiments are conducted on a single Linux machine with an AMD Ryzen Threadripper 3970X CPU @ 3.7GHz, 256 GB RAM, and a NVIDIA RTX TITAN GPU with 24GB of RAM.

References

1. Chen, L., Peng, J., Liu, Y., Li, J., Xie, F., Zheng, Z.: XBLOCK Blockchain Datasets: InPlusLab ethereum phishing detection datasets. http://xblock.pro/ethereum/ (2019)
2. Gong, L., Cheng, Q.: Exploiting edge features for graph neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9211–9219 (2019)
3. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in neural information processing systems. pp. 1024–1034 (2017)
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
5. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
6. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
7. Kumar, S., Zhang, X., Leskovec, J.: Predicting dynamic embedding trajectory in temporal interaction networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1269–1278 (2019)
8. Li, Z., Zhang, L., Song, G.: Gcn-lase: towards adequately incorporating link attributes in graph convolutional networks. arXiv preprint arXiv:1902.09817 (2019)
9. Liu, M., Liu, Y.: Inductive representation learning in temporal networks via mining neighborhood and community influences. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2202–2206 (2021)
10. Martins, A., Astudillo, R.: From softmax to sparsemax: A sparse model of attention and multi-label classification. In: International Conference on Machine Learning. pp. 1614–1623 (2016)
11. Mehran Kazemi, S., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., Brubaker, M.: Time2vec: Learning a vector representation of time. arXiv preprint arXiv:1907.05321 (2019)
12. 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)
13. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems. pp. 8024–8035 (2019)

14. Qu, L., Zhu, H., Duan, Q., Shi, Y.: Continuous-time link prediction via temporal dependent graph neural network. In: Proceedings of The Web Conference 2020. pp. 3026–3032 (2020)
15. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: Advances in neural information processing systems. pp. 1177–1184 (2008)
16. 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)
17. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3693–3702 (2017)
18. Sitzmann, V., Martel, J.N., Bergman, A.W., Lindell, D.B., Wetzstein, G.: Implicit neural representations with periodic activation functions. In: arXiv (2020)
19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
20. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
21. Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., et al.: Deep graph library: Towards efficient and scalable deep learning on graphs. arXiv preprint arXiv:1909.01315 (2019)
22. 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)
23. Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., Achan, K.: Inductive representation learning on temporal graphs. arXiv preprint arXiv:2002.07962 (2020)
24. Zhang, Z., Bu, J., Ester, M., Zhang, J., Yao, C., Li, Z., Wang, C.: Learning temporal interaction graph embedding via coupled memory networks. In: Proceedings of The Web Conference 2020. pp. 3049–3055 (2020)