Data Science and Engineering (2023) 8:98–111
https://doi.org/10.1007/s41019-023-00207-w

RESEARCH PAPER

Memory-Enhanced Transformer for Representation Learning on Temporal Heterogeneous Graphs

Longhai Li[1] · Lei Duan[1,3] · Junchen Wang[1] · Chengxin He[1] · Zihao Chen[1] · Guicai Xie[1] · Song Deng[2] · Zhaohang Luo[4]

Received: 4 December 2022 / Revised: 7 February 2023 / Accepted: 1 March 2023 / Published online: 22 March 2023

Abstract
Temporal heterogeneous graphs can model lots of complex systems in the real world, such as social networks and e-commerce applications, which are naturally time-varying and heterogeneous. As most existing graph representation learning methods cannot efficiently handle both of these characteristics, we propose a Transformer-like representation learning model, named THAN, to learn low-dimensional node embeddings preserving the topological structure features, heterogeneous semantics, and dynamic patterns of temporal heterogeneous graphs, simultaneously. Specifically, THAN first samples heterogeneous neighbors with temporal constraints and projects node features into the same vector space, then encodes time information and aggregates the neighborhood influence in different weights via type-aware self-attention. To capture long-term dependencies and evolutionary patterns, we design an optional memory module for storing and evolving dynamic node representations. Experiments on three real-world datasets demonstrate that THAN outperforms the state-of-the-arts in terms of effectiveness with respect to the temporal link prediction task.

Keywords Temporal heterogeneous graphs · Graph neural networks · Graph representation learning · Transformer

1 Introduction

Graph representation learning, as an important task in machine learning, has significant practical value in areas such as social networks and recommendation systems. Existing graph representation learning methods usually take static graphs as the input to obtain low-dimensional embeddings by encoding local non-Euclidean structures and have achieved extensive excellent performance in downstream tasks such as link prediction [1–3], node classification [4, 5], and graph classification [6, 7].

However, most graphs in the real world are naturally heterogeneous and dynamic, which cannot be accurately represented by static homogeneous graphs. Several studies incorporate heterogeneous data models into a unified graph model [8], promoting the research of graph data. Taking the example of a user-item interaction network in e-commerce scenarios, illustrated in Fig. 1a, there are two types of nodes (i.e., *user* and *item*) and three types of interactions (i.e., *favorite*, *browse* and *buy*). Additionally, each interaction is associated with a continuous timestamp to indicate when it occurred. In this paper, we define such interaction sequences

✉ Lei Duan
leiduan@scu.edu.cn

Longhai Li
lilonghai@stu.scu.edu.cn

Junchen Wang
wangjunchen@stu.scu.edu.cn

Chengxin He
hechengxin@stu.scu.edu.cn

Zihao Chen
chenzihao@stu.scu.edu.cn

Guicai Xie
guicaixie@stu.scu.edu.cn

Song Deng
dengsong@njupt.edu.cn

Zhaohang Luo
luozhaohang@qq.com

1 School of Computer Science, Sichuan University, Chengdu, China

2 Institute of Advanced Technology, Nanjing University of Posts & Telecommunications, Nanjing, China

3 Med-X Center for Informatics, Sichuan University, Chengdu, China

4 Nuclear Power Institute of China, Chengdu, China

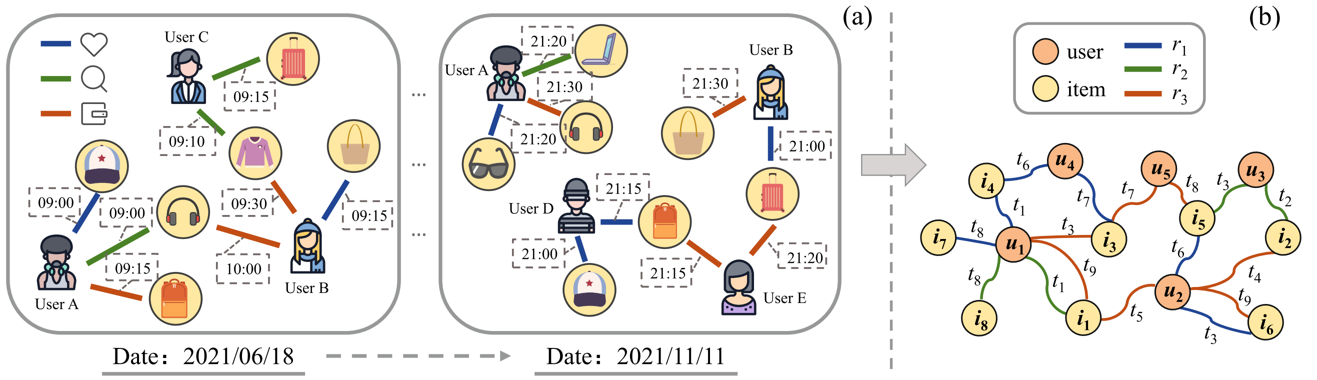

Fig. 1 A toy example of the temporal heterogeneous graph from a user-item interaction network. **a** User-item interaction network; **b** Temporal heterogeneous graph. Different colored lines represent different interactions, where the blue line denotes *favorite* (i.e., the heart icon), the green line denotes *browse* (i.e., the magnifier icon), and the orange line denotes *buy* (i.e., the wallet icon)

connecting different types of nodes as temporal heterogeneous graph (THG). It is of great significance to learn THG representations with dynamic and heterogeneous characteristics for modeling real-world complex systems.

Example 1 The user-item interaction network in an e-commerce scenario is illustrated in Fig. 1a. Two snapshots of the network are given for the dates of June 18, 2021 and November 11, 2021. There are two types of nodes (i.e., *user* and *item*) and three types of interactions (i.e., *favorite*, *browse* and *buy*), where *favorite* corresponds to the blue line, *browse* to the green line, and *buy* to the orange line. Additionally, each interaction is associated with a continuous timestamp to indicate the time it occurred. We can see that users' purchase intentions change dynamically over time.

In the case of the user-item interaction network shown in Fig. 1a, THG representation learning has the following challenges compared to static homogeneous graph representation learning:

- **(C1)** *How to model the heterogeneity*? The nodes and edges in THG are of various types and have rich semantics, making it difficult to obtain sufficient heterogeneous information just by encoding local graph structure.
- **(C2)** *How to model the continuous dynamics*? The edges in the THG are time-informed and time-dependent, i.e., each event occurs with a timestamp and current event may affect the occurrence of future events. For instance, there might be causal relationships between the interaction of searching for headphones on June 18 and the interaction of purchasing headphones on November 11 by user A. Therefore, both efficient methods of converting temporal information into dynamic features and temporal constraints are needed to avoid violating the temporal causality between interactions.
- **(C3)** *How to deal with new nodes*? The dynamics of the THG imply that new nodes will emerge in the future (e.g., users D and E are two new nodes that appeared on November 11 compared to June 18). In other words, these nodes are not present during training and many practical applications will require their embeddings to be generated in a timely manner. Therefore, it is necessary to construct an inductive modeling approach that generalizes the optimized representation to the new temporal subgraphs.

As for the heterogeneity, earlier methods [9, 10] preserve heterogeneous information by designing semantic meta-paths to generate heterogeneous sequences, and recent studies [2, 11–13] aggregate information from heterogeneous neighborhood by extending the message-passing process of graph neural networks (GNNs). Concerning dynamics, it is general to split temporal graphs into several static snapshots (i.e., discrete-time dynamic graph, DTDG [14]) and use RNNs or attention to capture the evolutionary patterns between snapshots [15–18]. Although these methods can learn graph dynamics of the THG to some extent, the temporal information within the same snapshot is usually ignored, and the scale of snapshots needs to be predetermined in advance. Recently, researchers have proposed continuous-time dynamic graph (CTDG [14]) approaches [19–24] to capture dynamics via passing information between different interactions, or using continuous-time functions to generate temporal embedding. In regard to the new nodes, inductive graph representation learning methods [5, 22, 23, 25] recognize structural features of node neighborhood by learning trainable aggregation functions, so that rapidly generate node embeddings in new subgraphs. Plenty of studies have attempted to solve the above challenges, nevertheless, few approaches can address them at the same time.

In this paper, we propose a novel Temporal Heterogeneous Graph Attention Network (THAN), which is a continuous-time THG representation learning method with Transformer-like attention architecture. To handle **C1**, we design a time-aware heterogeneous graph encoder to aggregate information from different types of neighbors. To handle **C2**, THAN samples temporally constrained neighbors and learns time-aware representation from historical heterogeneous events for a given node at any time point. It also encodes time information with a time encoder and incorporates them into the message propagation process. To handle **C3**, THAN can be thought of as a local aggregation operator based on neighbor sampling that recognizes the structural properties of a node's neighborhood and does not introduce global priori information.

THAN generates dynamic embeddings of nodes from their most recent neighbors. However, long-term dependencies and evolutionary patterns are not considered. Moreover, high-order information can be captured by stacking multiple THAN layers, but the cost is huge. To address these problems, we design an optional memory module to store and evolve the node representations. This module dynamically updates the node states as events occur and provides indirect access to distant neighbors by adding node memories to the raw inputs of THAN. The main contributions of our work are summarized as follows:

- We propose an inductive continuous-time THG representation learning method, which can capture both heterogeneous information and dynamic features.
- We introduce the dynamic transfer matrix and self-attention mechanism to implement the information aggregation of heterogeneous neighbors.
- We devise an optional memory module to enhance the representational ability of THAN by storing and updating the dynamic node states.
- We conduct experiments on three public datasets and the results demonstrate the superior performance of THAN over state-of-the-art baselines on the task of temporal link prediction.

The rest of the paper is organized as follows. We review related work in Sect. 2, and formulate the problem of temporal heterogeneous graph representation learning in Sect. 3. In Sect. 4, we discuss the critical techniques of THAN. We report a systematic empirical evaluation in Sect. 5, and conclude the paper in Sect. 6.

2 Related Work

Our work is related to representation learning on static graphs, temporal graphs (i.e., dynamic graphs), and self-attention mechanism on graphs.

Representation learning on static graphs Graph representation learning produces low-dimensional embeddings by modeling the topology and node attribute information. Early methods [9, 10, 26, 27] generate sequences of nodes by random walks among neighbors and then learn node co-occurrences to obtain representations. Luo et al. [28] define ripple distance over ripple vectors to optimize the walking procedure. In order to integrate rich node attribute features while learning network structure information, the GNN-based approaches [2, 4, 5, 11–13, 25] update node embeddings by aggregating neighborhood influence and propagating information across a multilayer network to capture the high-order patterns of the graph.

Focus on dealing with the heterogeneity, metapath2vec [9] and HIN2Vec [10] preserve heterogeneous information by designing semantic meta-paths, while heterogeneous GNNs [2, 11, 12] attempt to extend the message-passing procedure to handle different categories of information. Specifically, RGCN [2] introduces relation-specific transformations to encode features, HAN [12] designs hierarchical attention to describe node-level and semantic-level structures, HGT [11] uses meta-relation-based mutual attention to operate on heterogeneous graphs and learns implicit meta-paths. However, these methods cannot deal with temporal dynamics.

Representation learning on temporal graphs According to how temporal graphs are constructed, temporal graph representation learning methods can be divided into two categories: *discrete-time methods*, which describe the temporal graph as an ordered list of graph snapshots; *continuous-time methods*, which treat the temporal graph as an event stream with timestamps.

For the former, EvolveGCN [16] uses GCN to encode static graph structure and evolves the parameters of GCN by RNN. DySAT [17] uses structural attention to aggregate information from different neighbors in each snapshot and uses temporal attention to capture evolution over multiple snapshots. DyHATR [18] adopts hierarchical attention to learn heterogeneous information and applies RNNs with temporal attention to capture dependencies among snapshots. HTGNN [15] jointly models heterogeneous spatial and temporal dependencies through intra-relational, inter-relational, and cross-temporal aggregation. ROLAND [29] proposes a framework to extend static GNN to dynamic graphs. Although discrete-time methods succeed in learning the dynamic patterns of temporal graphs, they ignore time information within the same snapshot and lead to weakened connections between graph snapshots.

Recent studies [19–23, 30] have shown the superior performance of continuous-time methods in dealing with temporal graphs. JODIE [21] uses RNNs to propagate information in interactions and update node representations smoothly at different timesteps. TGAT [23] is designed as a GAT-like neural network, which propagates node information by sampling and aggregating historical neighbors, and learns high-order patterns by stacking multiple layers. TGN [24] proposes a general framework for encoding temporal graphs and captures long-term dependencies by preserving node states. CAW-N [22] proposes Causal Anonymous Walks (CAWs) to inductively represent a temporal graph and uses RNN to encode the walk sequences. These methods make full use of temporal information and model the dynamics of the graph without taking into account the heterogeneity. THINE [19] and HPGE [20] combine heterogeneous attention and Hawkes process to model graph heterogeneity and dynamics but do not consider the edge attributes.

Self-attention mechanism Transformer [31] proposed by Vaswani et al. for machine translation has achieved great success in NLP and CV tasks, which has recently been attempted for graph representation learning. For example, GTN [32] automatically generates useful meta-paths and learns new graph structures. Graphormer [33] generalizes positional encoding to the graph domain and uses scaled dot-product attention for message passing. Transformer relies on the self-attention mechanism to learn contextual information for sequences. A scaled dot-product attention layer can be defined as:

$$\mathrm{Attn}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d}}\right)\mathbf{V} \tag{1}$$

where $\mathbf{Q}$ denotes the 'queries', $\mathbf{K}$ the 'keys' and $\mathbf{V}$ the 'values'. They are the projections of the input $\mathbf{Z}$ on the matrices W_Q, W_K and W_V, where $\mathbf{Z}$ contains the node embeddings and their positional embeddings.

3 Preliminaries

In this section, we introduce the definition of temporal heterogeneous graph and the problem of temporal heterogeneous graph representation learning.

Definition 1 Temporal Heterogeneous Graph. A temporal heterogeneous graph is $\mathcal{G} = (V, E, T, \phi, \varphi)$, where V denotes the set of nodes corresponding to a node type mapping function $\phi : V \rightarrow A$, E denotes the temporal events (i.e., edges) corresponding to an event type mapping function $\varphi : E \rightarrow R$, and T denotes the set of timestamps. A and R are node type set and event type set, respectively, and $|A| + |R| > 2$.

Note that event $e = (u, v, t, \chi)$ means that there is an edge from u to v at time t, where χ denotes the edge feature and $r = \varphi(e)$ denotes the event type.

Example 2 In Fig. 1b, the temporal heterogeneous graph about user-item interactions consists of 13 nodes, 17 events (smaller subscript of t_i indicates that the event occurred earlier), two types of nodes, and three types of events. Specifically, $V = \{u_1, ..., u_5, i_1, ..., i_8\}$, $E = \{(u_1, i_1, t_1), ..., (u_5, i_5, t_8)\}$, $A = \{user, item\}$, $R = \{r_1, r_2, r_3\}$, $\phi(u) = user$, $\phi(i) = item$, r_1 denotes *favorite*, r_2 denotes *browse*, and r_3 denotes *buy*. According to the line color, we know that $\varphi(u_1, i_1, t_1) = r_2$ and $\varphi(u_5, i_5, t_8) = r_3$.

For any node pair (u, v), a *temporal causal path* is a set of events consisting of u as the source node of the start event and v as the target node of the terminal event. Therefore, the *temporal shortest path distance* $d_t(u, v)$ is defined as the minimum length of the temporal causal path from u to v with all events on the path occurring no later than t. Denote V_t as the set of nodes that appear up to time t, and for each node $v \in V_t$, define its k-hop *temporal neighbors* as:

$$\mathcal{N}_t^k(v) = \{u : d_t(u, v) \leq k, u \in V_t\} \tag{2}$$

For node v, we define its k-hop *temporal neighborhood* as $G_t^k(v)$, which is a subset of the temporal heterogeneous graph $\mathcal{G}$ and can be induced by $\mathcal{N}_t^k(v)$. $G_t^k(v)$ contains the source node v and its neighbors $\mathcal{N}_t^k(v)$, events between the nodes, and timestamps of these temporal events. The final representation of node v will generate relying on $G_t^k(v)$. Notice that we use $\mathcal{N}_t(v)$ and $G_t(v)$ to simplify the representation of $\mathcal{N}_t^1(v)$ and $G_t^1(v)$ in this paper, respectively.

Definition 2 Temporal Heterogeneous Graph Representation Learning. Given a temporal heterogeneous graph $\mathcal{G}$ and the node features X, it aims to learn a mapping function $\mathcal{F} : \mathcal{F}(\mathcal{G}, X) \rightarrow \mathbb{R}^{|V| \times d}$, where $|V|$ is the node size and d is the dimension of embeddings, $d \ll |V|$.

This mapping function maps nodes to low-dimensional vector space while preserving temporal, structural, and semantic information. For the sake of clarity, Table 1 summarizes the main notations used in this paper.

4 The Proposed Model

In this section, we present a Transformer-like graph attention architecture named THAN. It uses mapping matrices to project node embeddings into the same vector space, then passes neighborhood information by dot-product attention corresponding to different event types. Similar to GAT [5], THAN is designed as a local aggregation operator that captures high-order information by stacking multiple THAN layers. Figure 2 shows the architecture of the l-th THAN

Table 1 Summary of main notations

Notation	Description
$\mathcal{G}$	Temporal heterogeneous graph
V	Set of nodes
E	Set of edges
T	Set of timestamps
A	Set of node types
R	Set of edge types
ϕ	Node type mapping function
φ	Event type mapping function
$d_t(u, v)$	Temporal shortest path distance from u to v at time t
$\mathcal{N}_t^k(v)$	k-hop temporal neighbors of node v up to time t
$\mathbf{x}_v(t)$	Input embedding of node v
$\mathbf{e}$	Projection vector of event types
$\mathbf{n}$	Projection vector of node types
$\mathbf{M}$	Mapping matrix of meta relation $\langle \phi(u), \varphi(e), \phi(v) \rangle$
$\chi_{0,i}(t_i)$	Event feature between node v_0 and v_i at time t_i
W_Q, W_K, W_V	Projection matrices of 'query', 'key' and 'value'
$\tilde{\mathbf{h}}_0^l(t)$	Final output embedding of node v_0 at time t
$\mathbf{o}_i(t)$	Memory vector of node v_i at time t
$\mathbf{m}_i(t)$	Message of node v_i at time t

layer, which has three components: *temporal heterogeneous neighbor sampling*, *dynamic embedding mapping* and *temporal heterogeneous graph attention layer*. To capture long-term dependencies and evolutionary patterns, we design an optional memory module providing indirect access to distant neighbors, which dynamically updates the states of the nodes. After graph encoding, we use a heterogeneous graph decoder for the temporal link prediction task, which receives the node representations from THAN as inputs.

4.1 Temporal Heterogeneous Neighbor Sampling

For the purpose of improving the induction and generalization performance of the model, THAN does not select all but a certain number of neighbors from the temporal neighbors as the input. Given a node v_0 and time t, sample N neighbors from its 1-hop temporal neighbors $\mathcal{N}_t(v_0)$, denoted as $\{v_1, ..., v_N\}$.

We discuss two neighbor sampling strategies: *uniform random sampling*, where all temporal neighbors are randomly selected with equal probability; *top-N recent sampling*, where the time difference with the source node is calculated and sorted in ascending order, then select the top N neighbors. Intuitively, recent interactions reflect the node's current state better than distant interactions and have a greater influence on future events. On the contrary, the distant interactions may introduce noise. Therefore, we use the *top-N recent sampling* strategy to sample neighbors.

In the temporal heterogeneous graph, the number of different-typed events varies greatly, which can easily lead to an unbalanced distribution of the types of sampled neighbors. To avoid sampling bias as far as possible, THAN limits the number of samples of each event type to no more than M. If the total number of event types related to the source node is γ $(\gamma \le |R|)$, the total number of sampled neighbors N satisfies $N \le \gamma * M$.

4.2 Dynamic Embedding Mapping

For different nodes, TGAT [23] assumes that they are in the same feature distribution and share parameters of the

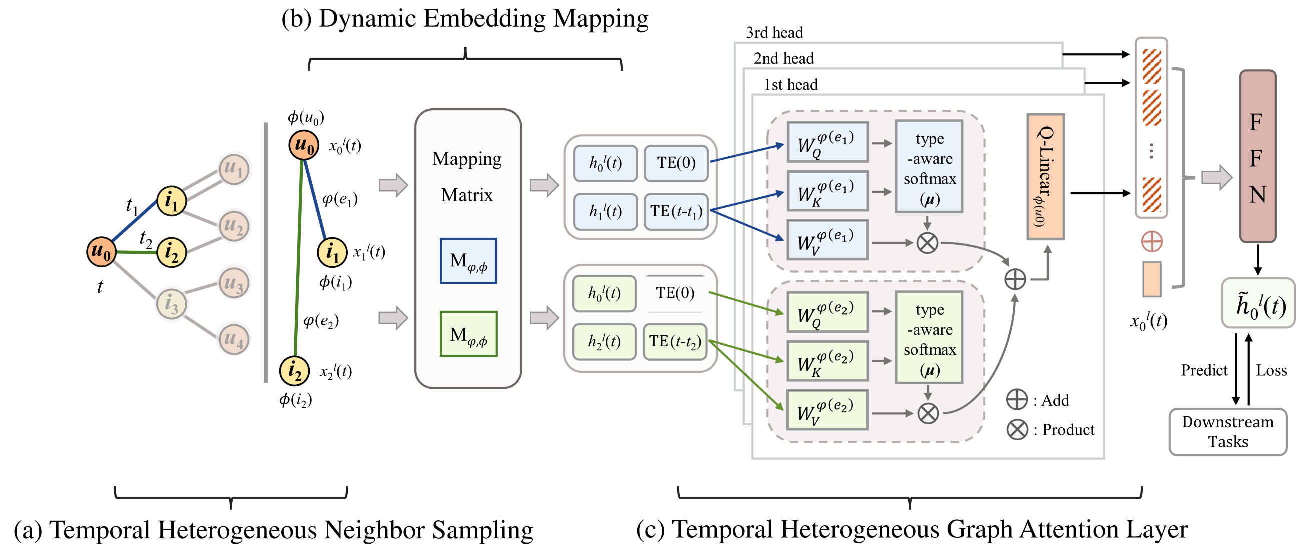

Fig. 2 The architecture of the l-th THAN layer for node u_0 at time t

model, which does not hold in heterogeneous graphs. Furthermore, in the real world, there might be multiple types of edges between two nodes, and different types may correspond to different vector distributions. So we must consider not only the diversity of nodes, but also edges when propagating information.

A straightforward solution is mapping features in different distributions to the same semantic space by transfer matrices. However, as the number of types increases, more parameters will be introduced into the model. To reduce the number of training parameters as well as to avoid large-scale matrix multiplication calculations, inspired by TransD [34], THAN projects node features from the node-type space to the event-type space by dynamically computing the transfer matrix with two type-related vectors.

Example 3 Suppose that there are two types of nodes, and three types of events in a heterogeneous graph. If there are four dimensions of each node type and each event type, respectively, we have to define the transfer matrix for each mapping from node type to event type. If we parameterize the transfer matrices directly, then the number of parameters is 96 (i.e., $2 \times 3 \times 4 \times 4$). However, the number of parameters is only 20 (i.e., $2 \times 4 + 3 \times 4$) if we use the projection vectors.

Given an event $e = (u, v, t)$ with its meta relation $\langle \phi(u), \varphi(e), \phi(v) \rangle$ [11], we define the dynamic mapping matrices as:

$$\mathbf{M}_{eu} = \mathbf{e}_{\varphi(e)} \mathbf{n}^{\top}_{\phi(u)} + \mathbf{I}^{d \times d} \tag{3}$$

$$\mathbf{M}_{ev} = \mathbf{e}_{\varphi(e)} \mathbf{n}^{\top}_{\phi(v)} + \mathbf{I}^{d \times d} \tag{4}$$

where $\mathbf{e}$ and $\mathbf{n}$ denote the projection vectors of event types and node types, respectively, both of which are trainable. The projected node embeddings are:

$$\mathbf{h}_u(t) = \mathbf{M}_{eu} \mathbf{x}_u(t) = \mathbf{n}^{\top}_{\phi(u)} \mathbf{x}_u(t) \mathbf{e}_{\varphi(e)} + \mathbf{x}_u(t) \tag{5}$$

$$\mathbf{h}_v(t) = \mathbf{M}_{ev} \mathbf{x}_v(t) = \mathbf{n}^{\top}_{\phi(v)} \mathbf{x}_v(t) \mathbf{e}_{\varphi(e)} + \mathbf{x}_v(t) \tag{6}$$

where $\mathbf{x}_u(t)$ and $\mathbf{x}_v(t)$ are the input embeddings of node u and v, respectively.

4.3 Temporal Heterogeneous Graph Attention Layer

Different events in a temporal heterogeneous graph may have different features, for example, in a question answering network, an answer interaction can be regarded as an event, and its features can be determined by the content. To enable event features to be propagated when aggregating information, THAN adds them to the node embeddings followed by a normalization layer (e.g., LayerNorm [35]). The event features will be resized to the same dimension as the node embeddings, and the output is:

$$\mathbf{z}_i(t_i) = \mathrm{LayerNorm}(\mathbf{h}^l_i(t_i) + \chi_{0,i}(t_i)) \tag{7}$$

where i indicates the i-th neighbor, $\chi_{0,i}(t_i)$ denotes the feature of event between node v_0 and v_i at time t_i. Here, we set $\chi_{0,0}(t)$ as zero vector.

Transformer [31] uses positional encoding to model relative position relationships, thus solving the problem that the attention mechanism cannot capture the sequential relationships between entities. In temporal graphs, a functional time encoder [36, 37] is usually used to map the time interval between nodes into a d_T-dimensional vector in place of positional encoding. THAN uses a Bochner-type functional time encoding [23, 37] as:

$$\mathrm{TE}(t) = \sqrt{\frac{1}{d_T}}[\cos(\omega_1 t), \sin(\omega_1 t), ..., \cos(\omega_{d_T} t), \sin(\omega_{d_T} t)] \tag{8}$$

where $\{\omega_i\}$s are learnable parameters. We merge the time embeddings with the node representations to obtain the node-temporal feature matrices as:

$$\mathbf{Zs}(t) = [\mathbf{z}^{e_1}_0(t) \| \mathrm{TE}(0), ..., \mathbf{z}^{e_N}_0(t) \| \mathrm{TE}(0)]^{\top} \tag{9}$$

$$\mathbf{Zn}(t) = [\mathbf{z}_1(t_1) \| \mathrm{TE}(t - t_1), ..., \mathbf{z}_N(t_N) \| \mathrm{TE}(t - t_N)]^{\top} \tag{10}$$

where $\mathbf{z}^{e_i}_0$ and $\mathbf{z}_i$ denote the mapped embeddings of the source node v_0 and its neighbor v_i corresponding to event e_i, respectively, and $\|$ denotes the 'concatenate' operation. **Zs** and **Zn** are forwarded to three different linear projections to obtain the 'query', 'key', and 'value':

$$\mathbf{Q} = \mathbf{Zs}(t) W^{\varphi(e_i)}_Q \tag{11}$$

$$\mathbf{K} = \mathbf{Zn}(t) W^{\varphi(e_i)}_K \tag{12}$$

$$\mathbf{V} = \mathbf{Zn}(t) W^{\varphi(e_i)}_V \tag{13}$$

where e_i denotes the event between v_0 and v_i, $W^{\varphi(e_i)}_Q$, $W^{\varphi(e_i)}_K$, and $W^{\varphi(e_i)}_V \in \mathbb{R}^{(d+d_T) \times d}$ denote the projection matrices. Due to the edge heterogeneity, the projection matrices cannot be shared directly, thus we use matrices of different types to distinguish different events while capturing the semantics of events. The attention weight α_i is given by:

$$\alpha_i = \frac{\exp(\beta_i)}{\sum_{j=1}^{N} \exp(\beta_j)} = \mathbf{Q}_i \mathbf{K}^{\top}_i \cdot \frac{\mu_{\phi(v_0), \varphi(e_i)}}{\sqrt{d}} \tag{14}$$

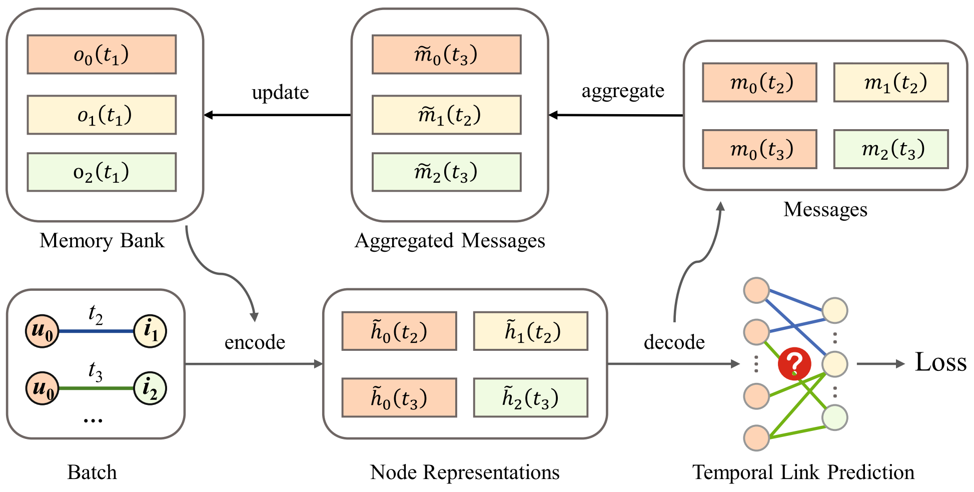

Fig. 3 Computation process of THAN with the memory module on a batch of events

and it reveals how v_i attends to the feature of v_0 through event e_i. In addition, not all types of events have the same contribution to the source node, so we set a learnable tensor $\mu \in \mathbb{R}^{|A|\times|R|}$ to adaptively adjust the scale of attention to different-typed events.

The self-attention aggregates the features of temporal neighbors and obtains the hidden representation for node v_i as $\alpha_i \mathbf{V}_i$, which can capture both node features and topological information. The next step is to map the representations back to the type-specific distribution of node v_0 so that they can be fused with the its features. We use a linear projection named Q-Linear to do this and the neighborhood representation is:

$$\mathbf{s}(t) = \sum_{i=1}^{N} \text{Q-Linear}_{\phi(v_0)}(\alpha_i \mathbf{V}_i) \tag{15}$$

To combine neighborhood representation with the source node feature, we concatenate and pass them to a feed-forward neural network as in TGAT [23]:

$$\tilde{\mathbf{h}}_0^l(t) = \text{FFN}(\mathbf{s}(t)\|\mathbf{x}_0^l(t)) \equiv \text{ReLU}([\mathbf{s}(t)\|\mathbf{x}_0^l(t)]W_0^l + b_0^l)W_1^l + b_1^l \tag{16}$$

Multi-head attention can effectively improve the model performance and stability. THAN can be easily extended to support a multi-head setup. Assuming the self-attention outputs come from P different heads, i.e., $s^i \equiv Attn^i(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, $i = 1, \ldots, P$. We first concatenate these neighborhood representations with the source node's feature and then carry out the same procedure in Eq. 16 as:

$$\tilde{\mathbf{h}}_0^l(t) = \text{FFN}(\mathbf{s}^1(t)\|\ldots\|\mathbf{s}^P(t)\|\mathbf{x}_0^l(t)) \tag{17}$$

where $\tilde{\mathbf{h}}_0^l(t) \in \mathbb{R}^d$ is the final output representation for node v_0 at time t, and it can be used for link prediction task with an encoder-decoder framework.

4.4 Memory Module

THAN stacks multiple networks to capture high-order patterns. However, as the number of layers increases, the cost of memory resources and training time grows exponentially. Moreover, constrained by the message-passing architecture, THAN cannot capture long-term features. To break this limitation, we devise a memory module (similar to TGN [24]) to save the historical states (i.e., memories) of the nodes. The states will be dynamically updated as events occur, thereby introducing long-term dependencies and indirectly accessing information from distant hops. Our experimental study, to be given in Sect. 5, demonstrates that the memory module costs less time than adding a THAN layer, but achieves similar or even superior performance.

Figure 3 shows the standard computation process of THAN with memory module on a batch of training data. It encodes the input data and the latest memory by THAN and output the node representations. The memory module uses the representations to compute messages and update node memory. However, the memory module does not directly affect the loss. To address this problem, we save messages of nodes involved in current batch at the end of training and update the memory with messages from previous batch before graph embedding. The memory module consists of the following components:

Memory Bank keeps the latest vector $o_i(t)$ for node v_i at time t, which is initialized as a zero vector. Its memory is updated on the occurrence of each event involving the node.

Message Function is a learnable function to compute a message $\mathbf{m}_i(t)$ for node v_i as follows:

$$\mathbf{m}_i(t) = \text{msg}(\tilde{\mathbf{h}}_i(t)\|\text{TE}(t - t_i^-)) \tag{18}$$

where $\tilde{\mathbf{h}}_i(t)$ is the representation from graph attention module, t_i^- is the time of the previous event involving node v_i, $msg(\cdot)$ is the message function and we use FFN in this paper.

In the case of an event $e_{ij}(t)$ between source node v_i and target node v_j, two messages (i.e., $m_i(t)$ and $m_j(t)$) can be computed. Different from TGN [24], we concatenate the node representation with the time embedding of the time span, while TGN considers both source node memory, target node memory, time embedding and edge features.

Message Aggregator is an aggregation function to aggregate messages generated from *message function* component. In batch processing node v_i may involve multiple events and each event corresponding to a message. These messages $\mathbf{m}_i(t_1), ..., \mathbf{m}_i(t_b)$ are aggregated with the following formulation:

$$\tilde{\mathbf{m}}_i(t) = \text{agg}(\mathbf{m}_i(t_1), ..., \mathbf{m}_i(t_b)) \tag{19}$$

where $t_1, ..., t_b \le t$, $agg(\cdot)$ can be optionally chosen as RNNs or attention networks. In this paper, we simply keep the *latest* message for a given node, which is considered from an efficiency perspective as it does not need to learn.

Memory Updater is used to update the memory of the nodes with the aggregated messages. Its formulation is:

$$\mathbf{o}_i(t) = \text{update}(\tilde{\mathbf{m}}_i(t), \mathbf{o}_i(t^-)) \tag{20}$$

where $update(\cdot)$ is a learnable memory update function like LSTM [38] or GRU [39]. When an interaction event happens, the memories of both nodes it involves will be updated.

4.5 Heterogeneous Graph Decoder

Heterogeneous graph decoder aims to reconstruct heterogeneous edges of the graph relying on the node representations, in other words, it scores edge triples through a function $\mathcal{H} : \mathbb{R}^d \times \mathbb{R}^{d_r} \times \mathbb{R}^d \to \mathbb{R}$, where d_r denotes the dimension of edge type embeddings. We compute node representations through a l-layer THAN encoder and use a feed-forward neural network as the scoring function, thus an event (u, v, t) of type r can be scored as:

$$\mathcal{H}(u, v, t, r) = \text{FFN}(\tilde{\mathbf{h}}^l_u(t) \| \mathbf{r}_r \| \tilde{\mathbf{h}}^l_v(t)) \tag{21}$$

where u and v denote the source and target node, respectively, $\mathbf{r} \in \mathbb{R}^{d_r}$ denotes the edge type embedding.

As in previous work [2, 23], we train the model with negative sampling. For each observed example, we change the target node to construct a nonexistent event as a negative sample, so the number of positive samples is the same as that of negative samples. We optimize the cross-entropy loss as:

$$\begin{aligned} \mathcal{L} = \frac{1}{|\varepsilon|} \sum_{(u,v,t,r,y)\in\varepsilon} & -y \log \sigma(\mathcal{H}(u, v, t, r)) \\ - \quad & (1 - y)\log(1 - \sigma(\mathcal{H}(u, v, t, r))) + \lambda\|\theta\|_2^2 \end{aligned} \tag{22}$$

where ε denotes the total set of positive and negative triples, σ denotes the logistic sigmoid function, y denotes the sample label and takes the value of 1 for positive samples and 0 for negative samples, θ denotes the model parameters and λ controls the L2 regularization.

Table 2 Statistics of the three public datasets

Dataset	Node types	#Nodes	#Event types	#Events	Time span
Movielens	User	943	5	100,000	7 months
	Movie	1682			
Twitter	User	304,691	3	563,069	188 days
MathOverflow	User	24,818	3	506,550	2,350 days

5 Experiments

In this section, we present the details of experiments including experimental settings and results. Firstly, we introduce the dataset, baselines, and parameter settings. Secondly, the performance comparisons are demonstrated in detail. Thirdly, we compare the effectiveness of different variants. Finally, we test the inductive capability of our proposed model.

5.1 Experimental Settings

5.1.1 Datasets

We evaluate our model on three public datasets: Movielens, Twitter, and MathOverflow. The statistics of these datasets are listed in Table 2.

- **Movielens**[1] is a dataset of user ratings of movies at different times collected from the MovieLens website. We select two types of nodes: user and movie. Regarding different ratings of movies as different types of events, a total of five types of events are obtained.
- **Twitter**[2] collects public data on three types of relationships (retweet, reply, and mention) between users from the US social network Twitter.
- **MathOverflow**[3] is from MathOverflow, a question and answer site for professional mathematicians. There are three relationships between users in this dataset: a user answered or commented on another user's question, and a user commented on an answer.

[1] https://grouplens.org/datasets/movielens/100k.

[2] http://snap.stanford.edu/data/higgs-twitter.html.

[3] http://snap.stanford.edu/data/sx-mathoverflow.html.

5.1.2 Baselines

To demonstrate the effectiveness, we compare THAN with ten popular graph representation learning methods, which can be divided into three groups: static graph embedding (DeepWalk [27], metapath2vec [9], GraphSAGE [25], GAT [5], RGCN [2], and HGT [11]), discrete-time dynamic graph embedding (DySAT [17] and DyHATR [18]), and continuous-time dynamic graph embedding (TGAT [23] and HPGE [20]). We use the implementations of static graph embedding methods provided in the PyTorch Geometric (PyG) package [40], and for other baselines, use the code submitted by the authors on GitHub. Besides, we ignore the heterogeneity for homogeneous methods and ignore the temporal information for static methods. For fairness, the same decoder declared in Sect. 4.5 is used for the downstream temporal link prediction task.

- **DeepWalk** and **metapath2vec**: They are random walk-based network embedding methods designed for static graphs.
- **GraphSAGE** and **GAT**: They are two inductive GNN methods for static homogeneous graphs, aggregating and updating node representations in the message passing framework.
- **RGCN** and **HGT**: They are two GNN methods for static heterogeneous graphs, where the former maintains a unique linear projection weight for each edge type while the latter uses mutual attention based on meta-relations to perform message passing on heterogeneous graphs.
- **DySAT**: A discrete-time temporal graph embedding method and we split graph snapshots with the guidance in the paper.
- **DyHATR**: A discrete-time THG embedding method that uses hierarchical attention to learn heterogeneous information and incorporates RNNs with temporal attention to capture evolutionary patterns.
- **TGAT**: A continuous-time temporal graph embedding method that aggregates historical neighbors by self-attention to obtain node representations.
- **HPGE**: A continuous-time THG embedding method that integrates the Hawkes process into graph embedding to capture the excitation of historical heterogeneous events to current events.

5.1.3 Parameter Settings

THAN was implemented in PyTorch. We split the training and test set as 8:2 according to time order. For a fair comparison, we use the default parameter settings of the baselines and set the embedding (i.e., node output embeddings, time embeddings, and event type embeddings) dimension d as 32, regularization weight λ as 0.01, and dropout rate as 0.1. We employ Adam as the optimizer with a learning rate of 0.001. We randomly initialize the node vector if the dataset does not provide node features, and similarly, initialize the event features as zero vectors. For DeepWalk, metapath2vec, GraphSAGE, GAT, RGCN, and HGT, we set the max training epochs as 500 and use an early stopping strategy with the patience of 50. For DySAT and DyHATR, we split datasets into 10 snapshots. For our THAN, we set the event embedding dimension as 16, the number of layers as 2, attention heads as 4, epochs as 20 (30 for Movielens), learning rate as 0.001 (0.0001 for Twitter), batch size as 800 (500 for Movielens), and the number of samples for each type of neighbors as 10 (8 for Movielens). The implementation of THAN is publicly available.[4]

5.2 Effectiveness Analysis

We conduct the temporal link prediction task to verify the effectiveness and efficiency, which asks if a type-r edge exists between two nodes at time t. We run all methods five times on three datasets and evaluate the average AUC (Area under the receiver operating characteristic curve) and AP (Average precision) scores. The overall results are shown in Table 3.

Obviously, THAN achieves the state-of-the-art performance in AUC metric on all three datasets. Although THAN does not outperform all other methods in AP metric, it also has a considerable performance (i.e., AP score achieves the SOTA result on Movielens and Twitter datasets and over 0.9 on MathOverflow dataset). For the Movielens dataset, dynamic graph embedding methods outperform the static graph embedding methods that ignore temporal information in both AUC and AP metrics, because the former learn temporal information in fine-grained contexts. Specifically, DySAT and DyHATR obtain performance improvements due to considering the changes of graph structure over time. TGAT, HPGE, and our THAN perform better than DySAT and DyHATR, this phenomenon shows that it is important to make full use of temporal information compared with simply preserving evolving structures between snapshots. For the other two datasets, the results of all methods show a similar trend, which means they probably have the same network patterns and temporal motifs. It may be related to the fact that they are both user activities datasets.

The GNN-based approaches achieve better performance than the random walk-based approaches since they capture much more useful information about the graph structure and the node features are utilized. The heterogeneous graph methods perform better than the homogeneous graph

[4] https://github.com/scu-kdde/HGA-THAN-2022.

Table 3 Overall performance comparison on temporal link prediction task

Dataset	Movielens		Twitter		MathOverflow	
Model	AUC	AP	AUC	AP	AUC	AP
DeepWalk	67.35±0.3	71.26±0.3	57.73±0.7	63.63±0.9	63.73±0.2	73.47±0.4
metapath2vec	68.43±0.2	71.82±0.2	66.29±0.3	74.67±0.2	72.59±0.9	81.13±1.1
GraphSAGE	72.34±0.4	75.94±0.4	76.88±3.2	85.10±2.1	83.48±2.4	89.20±3.4
GAT	71.81±1.3	73.23±2.1	83.69±3.3	89.23±2.6	83.79±3.5	90.01±3.7
RGCN	69.49±0.4	76.51±0.5	84.18±0.6	91.41±0.8	84.02±0.2	92.91±0.2
HGT	73.44±1.1	80.01±0.7	88.54±0.5	93.06±0.3	86.53±1.5	**93.88±1.4**
DySAT	73.13±0.4	72.1±0.3	83.03±0.3	86.89±0.2	83.12±0.3	85.84±0.1
DyHATR	80.21±0.7	77.54±1.3	79.73±0.1	81.78±0.4	75.22±0.1	78.21±0.2
TGAT	82.00±0.4	79.46±0.4	89.55±0.3	90.43±0.2	82.23±0.6	83.25±0.6
HPGE	85.25±0.1	82.16±0.2	73.55±0.1	73.91±0.1	81.12±0.2	82.61±0.2
THAN	**88.63±0.1**	**86.77±0.2**	**91.84±0.2**	**93.43±0.2**	**90.33±0.1**	90.62±0.2

All results are converted to a percentage by multiplying by 100, and the standard deviations computed over ten runs. The best and second-best results in each column are highlighted in bold font and underlined

methods which indicates that integrating semantic information can benefit graph representation. In terms of AUC score, THAN improves performance by 3.38%, 2.29% and 8.1% compared to TGAT on three datasets, respectively. It demonstrates the effectiveness of our proposed model. In summary, our proposed approach works for two main reasons: (1) it effectively extracts structural features and fine-grained temporal information; (2) it reasonably handles heterogeneity in the process of message passing and aggregation.

5.3 Ablation Study

To demonstrate the effectiveness of each component in THAN, we conduct ablation experiments by removing/replacing one specific component at a time. We rename them as: (1) THAN w/o time: removing time embeddings; (2) THAN w/o μ: removing event type attention weight; (3) THAN w/o Qlin: removing linear projection Q-Linear; (4) THAN r-uniform: using the *uniform random sampling* strategy instead of *top-N recent sampling*.

We report the results of the ablation study in Fig. 4, from which we have the following observations: (1) THAN outperforms the others with components removed in all metrics; (2) Time embedding plays an important role in temporal graph representation learning; (3) Setting different attention weights for different event types helps to learn heterogeneous semantic information; (4) More recent neighbors are more useful for extracting temporal dynamics and better reflect the current state of the source node; (5) It makes sense to keep the same feature space to fuse features from different nodes. Besides, it is noteworthy that removing the Q-Linear component did not have a significant impact on model performance on the Twitter and MathOverflow datasets, that is because both these datasets have only one type of node, and there is no need to consider the consistency of feature distribution across different types of nodes.

5.4 Parameter Sensitivity

To investigate the robustness of THAN and find the most suitable hyperparameters, we analyzed the effect of the number of neighbor samples and attention heads on three datasets shown in Fig. 5. For fairness, we select the number of neighbor samples from $\{4, 6, 8, 10\}$, the number of attention heads from $\{1, 2, 4, 6\}$, and the rest of the parameters remain the same as the experimental settings in Sect. 5.1.

On the one hand, Fig. 5a, b can lead to the following conclusion: the scores of AUC and AP improve as the number of neighbor samples increases, but on the Movielens dataset there is a decreasing trend instead, which may be caused by the dense connections between nodes. Sampling more neighbors may introduce more noise, resulting in smooth node representations. On the other hand, Fig. 5c, d shows that the number of attention heads affects the performance of the model. Multi-head attention helps to obtain different aspect representations from different subspaces, thus enhancing the expressiveness.

5.5 Effectiveness of Memory Module

In this section, we perform detailed studies on different instances of THAN focusing on the trade-off between accuracy and efficiency. The design of each variant is as follows: (1) THAN$_{l1}$: only using one graph attention layer as the graph encoder; (2) THAN$^{\dagger}_{l1}$: one-layer encoder with memory module; (3) THAN$_{l2}$: stacking two graph attention layers.

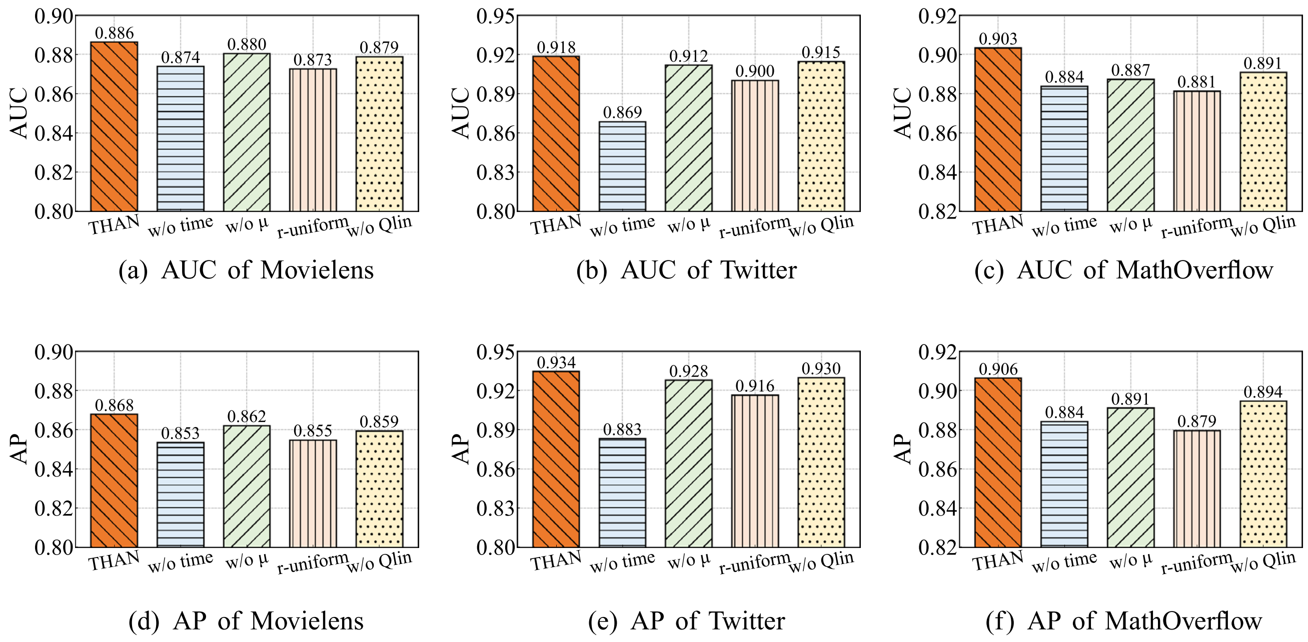

Fig. 4 Ablation study of THAN

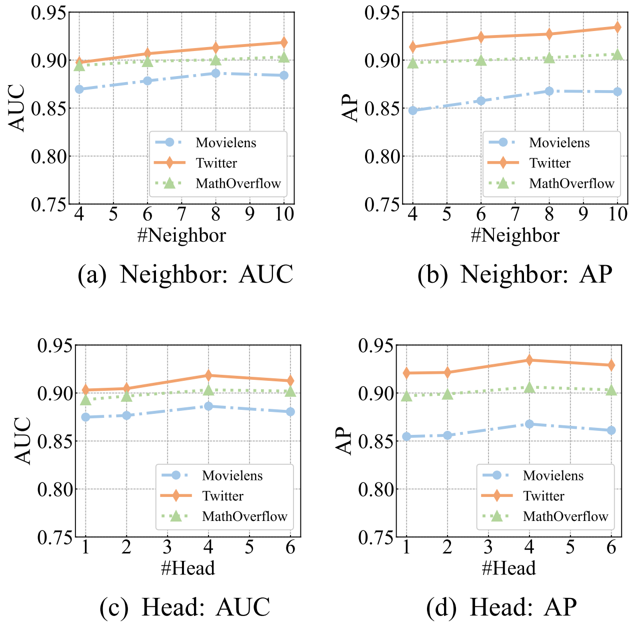

Fig. 5 Sensitivity analysis on the number of neighbor samples and attention heads

From Table 4, we can see that stacking two layers helps obtaining good performance (THAN$_{l2}$ vs THAN$_{l1}$), while the time costs increase by a factor of 8 to 21. Compared to adding a THAN layer, using memory module achieves similar or even superior model performance, but spends much less time (THAN$_{l2}$ takes about 13 times, 4 times and 19 times longer than THAN$^{\dagger}_{l1}$ on three datasets, respectively). On the one hand, the number of neighbors that need to be aggregated increases exponentially by adding a layer. On the other hand, when accessing the memory of the source node and its 1-hop neighbors, THAN is introducing long-term dependencies and indirectly accessing information from distant hops.

5.6 Inductive Capability Analysis

We further discuss the inductive performance of THAN with the same settings as TGAT, i.e., mask 10% of the nodes from the training set and predict the existence of future events containing these masked nodes. In this paper, we choose GraphSAGE, GAT and TGAT as the comparison models. They are proposed as inductive representation learning methods on graphs, and their inductive capabilities are demonstrated experimentally. Experiments were conducted on three datasets and the results are shown in Table 5. Intuitively, THAN outperformed the baselines in two metrics on all datasets, which demonstrates the inductive capability of THAN.

Table 4 Trade-off between accuracy (test AUC score in %) and speed (time cost per epoch in second) for different instances of the THAN

Dataset	Movielens		Twitter		MathOverflow	
Model	AUC	Time	AUC	Time	AUC	Time
THAN_{l1}	83.90±0.2	35.8±2.0	87.80±0.5	192.6±4.0	87.40±0.1	185.4±3.1
$\text{THAN}^{\dagger}_{l1}$	87.87±0.3	40.5±2.6	92.79±0.5	360.7±28.1	92.93±0.3	210.8±12.5
THAN_{l2}	88.63±0.1	524.2±5.4	91.84±0.2	1545.5±20.1	90.33±0.1	4046.8±19.7

All experiments are conducted in batch size of 200

Table 5 Results of inductive learning task

Dataset	Movielens		Twitter		MathOverflow	
Model	AUC	AP	AUC	AP	AUC	AP
GraphSAGE	70.58±0.8	71.37±1.6	73.15±3.9	78.44±3.5	70.56±5.5	74.23±5.2
GAT	69.48±1.2	72.02±1.9	75.77±4.4	80.61±5.1	71.73±4.8	76.42±5.6
TGAT	78.35±0.4	76.97±0.3	85.87±0.3	88.61±0.3	74.14±0.5	75.96±0.4
THAN	82.71±0.2	80.67±0.2	88.41±0.2	90.69±0.3	80.93±0.3	80.52±0.3

6 Conclusion

Existing graph representation learning methods cannot well capture the information of temporal heterogeneous graphs. This paper proposes the THAN, which is a continuous-time temporal heterogeneous graph representation learning method. THAN uses dynamic transfer matrices to map different-typed nodes to the same feature space and aggregates neighborhood information based on the type-aware self-attention mechanism. To efficiently utilize temporal information, THAN uses a functional time encoder to generate time embeddings that are naturally integrated into the neighbor aggregation process. THAN is an inductive message-passing model based on historical neighbor sampling that not only captures temporal dynamics but also efficiently extracts topological features. In addition, we devise an optional memory module to store node states and capture long-term dependencies. It improves the model performance and takes less time than stacking a new THAN layer. The experimental results on three public datasets demonstrate that THAN outperforms the baselines on the temporal link prediction task.

In future work, on one hand, we plan to explore the usage of THAN in various fields, such as recommender systems, social networks, and biological interaction networks. On the other hand, we try to understand the specific patterns/motifs of temporal heterogeneous networks from different domains. Furthermore, the large-scale temporal heterogeneous graph embedding is another direction worthy of further investigation.

Author Contributions Longhai Li proposes the methodology, completes the experiments and writes the manuscript. Lei Duan provides instructions and revises the manuscript. Junchen Wang and Zihao Chen assist in improving the experiment. Junchen Wang, Chengxin He, Guicai Xie, Song Deng and Zhaohang Luo jointly help to write the manuscript.

Funding This work was supported in part by the National Natural Science Foundation of China (61972268) and the Joint Innovation Foundation of Sichuan University and Nuclear Power Institute of China.

Availability of data and materials The datasets used in experiments can be downloaded from the following URLs: Movielens: https://grouplens.org/datasets/movielens/100k Twitter: http://snap.stanford.edu/data/higgs-twitter.html MathOverflow: http://snap.stanford.edu/data/sx-mathoverflow.html.

Declarations

Conflicts of interest We would like to submit the enclosed manuscript entitled "Memory-Enhanced Transformer for Representation Learning on Temporal Heterogeneous Graphs", which we wish to be considered for publication in Data Science and Engineering. No conflict of interest exits in the submission of this manuscript, and manuscript is approved by all authors for publication. I would like to declare on behalf of my co-authors that the work described was original research that has not been published previously, and not under consideration for publication elsewhere, in whole or in part. All the authors listed have approved the manuscript that is enclosed.

References

1. Kipf TN, Welling M (2016) Variational graph auto-encoders. CoRR arXiv:1611.07308
2. Schlichtkrull MS, Kipf TN, Bloem P, van den Berg R, Titov I, Welling M (2018) Modeling relational data with graph convolutional networks. In: Proceedings of the 15th international conference on semantic web, vol 10843, pp 593–607
3. He C, Duan L, Zheng H, Li-Ling J, Song L, Li L (2022) Graph convolutional network approach to discovering disease-related CIRCRNA–MIRNA–MRNA axes. Methods 198:45–55
4. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th international conference on learning representations
5. Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: Proceedings of the 6th international conference on learning representations
6. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. In: Proceedings of the 34th international conference on machine learning, vol 70, pp 1263–1272
7. Ying Z, You J, Morris C, Ren X, Hamilton WL, Leskovec J (2018) Hierarchical graph representation learning with differentiable pooling. In: Proceedings of the 32nd international conference on neural information processing systems, pp 4805–4815
8. Tuteja S, Kumar R (2022) A unification of heterogeneous data sources into a graph model in e-commerce. Data Sci Eng 7:57–70
9. Dong Y, Chawla NV, Swami A (2017) metapath2vec: scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 135–144
10. Fu T, Lee W, Lei Z (2017) Hin2vec: explore meta-paths in heterogeneous information networks for representation learning. In: Proceedings of the 2017 ACM on conference on information and knowledge management, pp 1797–1806
11. Hu Z, Dong Y, Wang K, Sun Y (2020) Heterogeneous graph transformer. In: Proceedings of the 29th international conference on world wide web, pp 2704–2710
12. Wang X, Ji H, Shi C, Wang B, Ye Y, Cui P, Yu PS (2019) Heterogeneous graph attention network. In: Proceedings of the 28th international conference on world wide web, pp 2022–2032
13. Zhao J, Wang X, Shi C, Hu B, Song G, Ye Y (2021) Heterogeneous graph structure learning for graph neural networks. In: Proceedings of the 35th AAAI conference on artificial intelligence, pp 4697–4705
14. Kazemi SM, Goel R, Jain K, Kobyzev I, Sethi A, Forsyth P, Poupart P (2020) Representation learning for dynamic graphs: a survey. J Mach Learn Res 21:70–17073
15. Fan Y, Ju M, Zhang C, Ye Y (2022) Heterogeneous temporal graph neural network. In: Proceedings of the 2022 SIAM international conference on data mining, pp 657–665
16. Pareja A, Domeniconi G, Chen J, Ma T, Suzumura T, Kanezashi H, Kaler T, Schardl TB, Leiserson CE (2020) Evolvegcn: evolving graph convolutional networks for dynamic graphs. In: Proceedings of the 34th AAAI conference on artificial intelligence, pp 5363–5370
17. Sankar A, Wu Y, Gou L, Zhang W, Yang H (2020) Dysat: deep neural representation learning on dynamic graphs via self-attention networks. In: Proceedings of the 13th international conference on web search and data mining, pp 519–527
18. Xue H, Yang L, Jiang W, Wei Y, Hu Y, Lin Y (2020) Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal RNN. In: Proceedings of the 2020 European conference on machine learning and knowledge discovery in databases, vol 12457, pp 282–298
19. Huang H, Shi R, Zhou W, Wang X, Jin H, Fu X (2021) Temporal heterogeneous information network embedding. In: Proceedings of the 30th international joint conference on artificial intelligence, pp 1470–1476
20. Ji Y, Jia T, Fang Y, Shi C (2021) Dynamic heterogeneous graph embedding via heterogeneous hawkes process. In: Proceedings of the 2021 European conference on machine learning and knowledge discovery in databases, vol 12975, pp 388–403
21. Kumar S, Zhang X, Leskovec J (2019) Predicting dynamic embedding trajectory in temporal interaction networks. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining, pp 1269–1278
22. Wang Y, Chang Y, Liu Y, Leskovec J, Li P (2021) Inductive representation learning in temporal networks via causal anonymous walks. In: Proceedings of the 9th international conference on learning representations
23. Xu D, Ruan C, Körpeoglu E, Kumar S, Achan K (2020) Inductive representation learning on temporal graphs. In: Proceedings of the 8th international conference on learning representations
24. Rossi E, Chamberlain B, Frasca F, Eynard D, Monti F, Bronstein MM (2020) Temporal graph networks for deep learning on dynamic graphs. CoRR arXiv:2006.10637
25. Hamilton WL, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the 31st international conference on neural information processing systems, pp 1024–1034
26. Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 855–864
27. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining, pp 701–710
28. Luo J, Xiao S, Jiang S, Gao H, Xiao Y (2022) ripple2vec: node embedding with ripple distance of structures. Data Sci Eng 7:156–174
29. You J, Du T, Leskovec J (2022) ROLAND: graph learning framework for dynamic graphs. In: Proceedings of the 28th ACM SIGKDD international conference on knowledge discovery and data mining, pp 2358–2366
30. Trivedi R, Farajtabar M, Biswal P, Zha H (2019) Dyrep: learning representations over dynamic graphs. In: Proceedings of the 7th international conference on learning representations
31. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Proceedings of the 31st international conference on neural information processing systems, pp 5998–6008
32. Yun S, Jeong M, Kim R, Kang J, Kim HJ (2019) Graph transformer networks. In: Proceedings of the 33rd international conference on neural information processing systems, pp 11960–11970
33. Ying C, Cai T, Luo S, Zheng S, Ke G, He D, Shen Y, Liu T (2021) Do transformers really perform badly for graph representation? In: Proceedings of the 35th international conference on neural information processing systems, pp 28877–28888
34. Ji G, He S, Xu L, Liu K, Zhao J (2015) Knowledge graph embedding via dynamic mapping matrix. In: Proceedings of the 53rd annual meeting of the association for computational linguistics, pp 687–696
35. Ba LJ, Kiros JR, Hinton GE (2016) Layer normalization. CoRR arXiv:1607.06450
36. Kazemi SM, Goel R, Eghbali S, Ramanan J, Sahota J, Thakur S, Wu S, Smyth C, Poupart P, Brubaker M (2019) Time2vec: learning a vector representation of time. CoRR arXiv:1907.05321
37. Xu D, Ruan C, Körpeoglu E, Kumar S, Achan K (2019) Self-attention with functional time representation learning. In: Proceedings

of the 33rd international conference on neural information processing systems, pp 15889–15899
38. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
39. Cho K, van Merrienboer B, Gülçehre Ç, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2004 conference on empirical methods in natural language processing, pp 1724–1734
40. Fey M, Lenssen JE (2019) Fast graph representation learning with pytorch geometric. CoRR arXiv:1903.02428