Applied Intelligence (2024) 54:7151–7169
https://doi.org/10.1007/s10489-024-05548-1

Multi-task recommendation based on dynamic knowledge graph

Minwei Wen[1] · Hongyan Mei[1] · Wei Wang[2] · Xiaorong Xue[1] · Xing Zhang[1]

Accepted: 19 May 2024 / Published online: 3 June 2024

Abstract
Introducing knowledge graphs into recommender systems effectively solves sparsity and cold start problems. However, existing KG recommendation methods such as MKR mostly rely on static knowledge graphs, ignoring that nodes and edges in the graph dynamically change over time, leading to problems such as insufficient timeliness, inability to describe context dependencies, data redundancy, and noise. We propose a multi-task learning method for recommendation enhancement based on dynamic knowledge graphs, MTRDKG, which models the dynamic knowledge graph as a series of continuous time events. Specifically, after node events (node-level or interactions between nodes) occur, the memory state of the nodes is updated through a temporal graph network (TGN), and node temporal embeddings are generated to capture the nodes' attributes, contextual relationships, and dynamic change information. We use the node embeddings generated by TGN in conjunction with the recommended items as a shared part, aiming to integrate dynamic knowledge graph information into the recommendation task, thereby improving the recommendation effect. Extensive experiments were conducted with four real-world datasets and state-of-the-art baseline methods. The results show that MTRDKG outperforms existing methods in terms of recommendation accuracy and knowledge graph embedding quality, especially in dealing with datasets of different sparsity levels.

Keywords Recommender systems · Knowledge graphs · Multi-task learning · Temporal graph network

1 Introduction

Recommender systems (RS) are intelligent systems designed to address information overload and provide personalized recommendations, whereas collaborative filtering (CF) algorithms are among the most commonly used recommendation technologies. However, CF algorithms have inherent issues such as cold starts and sparsity. To address these problems, some researchers have attempted to introduce side information into RS, e.g., social networks, item attributes, and multimedia. Knowledge graphs (KG) organize entities and relationships in the real world graphically, capable of efficiently expressing the deep semantic connections between entities [1]. Moreover, KG introduces semantic relevance between items, which can capture users' personalized preferences more accurately by deeply mining the associations between entities [2]. Therefore, integrating KG into RS has become a research direction of great interest. Compared to traditional RS, knowledge graph-based recommender systems (KGRS) endow the reasoning process with more excellent usability, making the recommendation mechanism more transparent and enhancing the explainability of recommendations [3].

Existing KGRS can generally be divided into two categories: path-based methods and embedding-based methods [4]. (1) Path-based methods enhance recommendation effects by constructing user-item graphs and utilizing the connection patterns of entities within the graph. Although path-based recommendation methods can intuitively use KG, they are

✉ Hongyan Mei
liaoning_mhy@126.com

Minwei Wen
w18779721623@163.com

Wei Wang
wangwei03770377@126.com

Xiaorong Xue
xr_986@163.com

Xing Zhang
dr_zhangxing@163.com

[1] School of Electronics and Information Engineering, Liaoning University of Technology, Jinzhou 121001, Liaoning, China

[2] School of Electrical Engineering, Liaoning University of Technology, Jinzhou 121001, Liaoning, China

limited by manually designed meta-paths and cannot automatically discover and reason about potential connection patterns. (2) Embedding-based methods typically directly adopt information from KG to enhance the feature representations of items or users. This approach is one of the most commonly used methods in designing KGRS. Effectively integrating KG information relies on knowledge graph embedding (KGE) techniques, which are aimed at mapping entities and relationships in KG into a low-dimensional vector space while preserving the inherent structure of KG. For example, CKE [5] uses TransR [6] and autoencoders to encode the structural knowledge and textual features of items to enrich item representations. Similarly, JointE [7] utilizes bidirectional convolutional operations to promote interactions between entities and relationships to fully capture KG's potential knowledge, while SAttLE [8] employs many self-attention heads as the key for query-dependent projection to capture the interaction information between entities and relationships. However, they use different scoring functions to measure the plausibility of facts, e.g., CKE uses a distance-based scoring function to measure the plausibility of a fact based on the distance between two entities, and JointE and SAttLE uses a similarity-based scoring function to output the probability of a fact triad through a neural network using the embeddings of the entities and the relationships as the input [9]. These methods are all built on practical KGE foundations. However, KG has incompleteness [10] and potential errors, and directly applying vectors obtained from KGE techniques may negatively affect the recommendation results.

Given the relevance and complementarity between RS and KGE tasks, some researchers utilize multi-task learning (MTL) methods to exploit their interrelations fully, utilizing the useful information in the two related tasks to help improve the generalization performance of all tasks [11]. For example, Wang et al. [12] assist the recommendation task with the KGE task, optimizing the recommendation task by alternately learning the two tasks through the relevancy between items in the interaction graph and entities in the knowledge graph. However, existing methods like MKR introduce a static knowledge graph (SKG) as side information, leading to several issues: (1) Insufficient timeliness and inability to reflect changes in user preferences. (2) Node and relationship representations depend on context, and SKG cannot dynamically adjust in different contexts. (3) Data redundancy and noise, increasing the complexity of data processing.

Addressing the issues in previous works, we propose MTRDKG, a multi-task learning method enhanced by dynamic knowledge graphs (DKG) for recommendations. Introducing DKG into RS is a novel and promising research direction. Notably, various extension methods can bring positive effects. For instance, in integrated systems [13, 14], closely combining DKG with recommendation models can enhance the models' ability to capture time sensitivity. In two-stage systems [15], the DKG can provide rich contextual information to the RS in the first stage, and this information is used in the second stage to assist in making more accurate recommendation decisions. However, we choose to implement DKG under the MTL framework because it can better adapt to the characteristics between different tasks through mechanisms such as sharing model parameters, joint training, and iterative optimization. The novelty of MTRDKG lies in the introduction of TGN [16] to capture the temporal changes of DKG. In response to question (1), the memory structure in TGN can store the historical representations of nodes and dynamically update over time, effectively capturing the long-term dependencies between nodes. In response to question (2), the graph neural network (GNN) layer serves two functions: first, the message-passing function generates information from the graph's dynamic interactions, encoding the relationships and interaction characteristics between nodes. second, the message aggregation function aggregates this generated information into a comprehensive node representation, effectively capturing the graph's temporal changes and complexity. In response to question (3), Temporal attention allocates different weights to messages from different times, better capturing cross-temporal dependencies. Furthermore, the updater and embedding module in TGN can integrate messages into the current node representation to update node embedding representations, which are then used to replace the entity representations in MKR as the shared part. Experiments show that exploring the dynamic evolution patterns of KG can enhance recommendation performance. Additionally, sharing item knowledge facilitates training in dynamic knowledge graph embedding (DKGE) tasks.

The main contributions of this paper are as follows:

1. We propose a multi-task learning method enhanced by dynamic knowledge graphs for recommendations, MTRDKG, which models DKG as a series of continuous-time events, better reflecting the changes in the real world. This is the first time DKG has been integrated under the MTL framework to enhance the performance of recommender systems.
2. We use node embeddings to replace the entities in MKR as the shared part, which includes the nodes' features and encompasses information about the nodes' temporality, neighboring nodes, and edge attributes in DKG. This approach can more comprehensively reflect the characteristics of nodes in DKG, achieving more accurate recommendations.
3. We conducted experiments on CTR prediction, Top-K recommendation, and future link prediction on four real datasets. The experimental results show that MTRDKG outperforms the most advanced baseline methods in performance.

2 Related work

Our proposed method jointly trains recommendation tasks and DKGE tasks under the MTL framework. Therefore, this section will briefly review the three most relevant aspects: DKGE methods, MTL, and MTL-based KGRS.

2.1 Dynamic knowledge graph embedding methods

Based on the methods of processing temporal information, DKGE methods can be divided into two categories: embedding snapshot graphs [17] and modeling temporal evolution [18].

Embedding snapshot graph methods decompose the dynamism of KG into a series of static snapshot graphs, then use KGE techniques to learn the evolutionary knowledge of snapshot graphs at different time points. For example, DynamicTriad [19] captures the structural information of each snapshot graph and the continuity of the neighboring snapshot graphs' embedding representations by simulating the triadic closure process. Dyngraph2vec [20] combines multi-layer nonlinear networks to analyze the structural features of each snapshot and uses recurrent network layers to reveal the temporal evolution between snapshots. Compared to DynamicTriad, Dyngraph2vec is more effective in capturing longer-term evolution patterns. However, as the number of snapshot graphs increases, the scalability of these two methods deteriorates. To address scalability issues, some works attempt to learn the periodic patterns of nodes from snapshot graphs. For instance, Pikachu [21] uses a temporal random walk strategy to capture the network topology and fine-grained temporal information. Although this method performs well in dynamic network anomaly detection, it converges slowly. DySAT [22] combines structural and temporal self-attention mechanisms, effectively learning DKG's local structures and temporal evolution. However, DySAT is sensitive to the time window size setting.

Modeling temporal evolution directly embeds temporal information in DKG to capture the temporal changes of nodes and edges dynamically. For example, CTDNE [23] and LSTM-Node2vec [24] adopt a time-sensitive random walk strategy to generate continuous and consistent node embeddings. They reveal the temporal sequential relationships between nodes but ignore the temporal evolution process of specific node neighborhoods. Based on this, HTNE [25] considers the neighborhood sets of nodes at different time points through the Hawkes process and uses an attention mechanism to determine the influence of the node's historical neighborhoods. However, HTNE requires substantial computational resources. Unlike the methods mentioned above focused on the temporal evolution of node embeddings, EvolveGCN [26] focuses on the evolution of the graph model structure (i.e., network parameters), making it more sensitive to changes in graph morphology. However, EvolveGCN highly depends on the quality and granularity of temporal data. TGAT [27] is an inductive representation learning model based on temporal graphs, capable of efficiently processing unknown nodes. Nevertheless, TGAT faces the issue of information loss for exponentially expanded and invisible neighbors. TGNs [16] is a general model based on transfer learning. Specifically, TGAT is a particular case of TGNs when the memory module in TGNs is absent. This paper selects TGNs as the foundation for DKG, mainly because TGNs are particularly suitable for handling time-sensitive data. Secondly, TGNs use an event-driven message-passing mechanism that can reflect the latest interactions and relationships between nodes. Lastly, TGNs are a general framework that can be flexibly applied in the recommendation domain.

2.2 Multi-task learning

MTL aims to improve model generalization ability by utilizing domain-specific information in the training signals of related tasks, with the core idea of leveraging the commonalities and differences between different tasks to learn more generalized and robust feature representations [28]. However, potentially harmful interactions between tasks can lead to the issue of negative transfer, necessitating the selection of an appropriate parameter-sharing mechanism based on task relevance. MTL employs two common sharing mechanisms: hard parameter sharing and soft parameter sharing [29]. Under hard parameter sharing, lower-level parameters are fully shared, while the top layers are trained independently for specific tasks. However, hard sharing is sensitive to task relevance and unsuitable for improving the performance of multiple tasks with low relevance [30]. On the other hand, soft sharing establishes independent models for each task, allowing for the sharing of some helpful information between tasks through relevance weights, such as attention mechanisms. Compared to hard sharing, soft sharing is unaffected by task relevance and offers greater flexibility in parameter sharing. The MTRDKG model proposed in this paper adopts a soft-sharing mechanism. RS and DKGE tasks have independent models and parameters, learning high-order interaction features between tasks through information-sharing units. Notably, due to differences in data characteristics, goals, and optimization methods between RS tasks and KGE tasks, most current MTL-based KGRS research opts for soft sharing mechanisms [31–33].

2.3 MTL-Based KGRSs

In MTL-based KGRS, recommended items and one or multiple nodes in KG are associated, sharing similar neighborhood

structures in RS and KGE tasks, which can share similar features in a low-level or non-task-specific latent space [12]. However, traditional KGRS methods directly adopt KGE vectors to represent recommended users and items or associate RS tasks with KGE tasks through simple summation operations. Due to differences between tasks or data, this approach may impair the effect of recommendation. MKR [12] designed a cross-compression unit to adaptively extract shared features between RS and KGE tasks through cross-modal feature learning, a soft parameter-sharing mechanism [34]. However, this unit uses an outer product for cross-multiplication and then a one-dimensional vector parameter for compression in joint training, leading to insufficient learning capacity for interaction features. Based on this, CAKR [35] designed a cross-attention unit that optimizes interactions between items and entities by integrating attention mechanisms. However, MKR and CAKR share three common issues: (1) RS tasks cannot fully capture user history information. (2) KGE tasks cannot effectively utilize deep multi-relational semantic information. (3) Inability to effectively handle datasets with varying degrees of sparsity. To address these issues, EMKR [36] made improvements: firstly, using attention mechanisms in RS tasks to aggregate user historical behaviors, accurately capturing user interests. Secondly, relation-aware graph convolutional networks in KGE tasks are used to capture deep multi-relational domain features. Thirdly, a two-stage training strategy should be adopted to handle datasets with different sparsity levels. However, the above three models only introduce KG as a single type of side information, reinforcing item attributes but not considering user relationships. TMKG [37] utilizes MTL to integrate two types of edge information from trust graphs and KG in an end-to-end manner, exploring the fine-grained implicit relationships between external heterogeneous graphs. To address the issues of generality and robustness in existing methods, Multi-Rec [38] uses cross units to separately learn the feature information and structural information of users and KG, then uses exchange units to learn the association information between tasks. It should be noted that, unlike methods [12, 35–38], which introduce SKG as side information, MTRDKG will dynamically update KG.

3 Preliminary

Static graph SKG is usually represented by $\mathcal{G} = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of nodes, and $E \in V \times V$ is the set of edges. Each node and each edge have attributes v_i and e_{ij} respectively, and $i, j = 1, 2, \ldots, n$. A graph neural network (GNN) aims to update the current node's embedding z_i by aggregating the embedding information of neighboring nodes, denoting data transfer from neighborhood node i to node j. The transfer process is subject to the principle of local consistency.

$$m_{ij} = MSG(v_i, v_j, e_{ij}) \tag{1}$$

$$z_i = AGG(\{m_{ij} | j \in \varphi_i\}, v_i) \tag{2}$$

where $\varphi_i = \{j : (i, j) \in E\}$ denotes the set of neighbourhoods of i, $MSG(\cdot)$and $AGG(\cdot)$ are learnable information transfer and aggregation functions, respectively.

Dynamic graphs We model the DKG as a collection of events $\mathcal{G} = \{X(t) | t \in (t_1, t_2, \ldots, t_n)\}$ based on a sequence of time. The events reflect the structure and attribute changes of the KG at time $0 \leq t_1 \leq t_2 \leq, \ldots, \leq t_n$. They can generally be divided into two categories: (1) A node-wise event is represented by $v_i(t)$. Here, i and v denote the node's index and attribute value. When the event $v_i(t)$ has occurred before time t, it can be deleted or updated; otherwise, it is created. (2) Node interaction events are represented by $e_{ij}(t)$, involving relationships between two or more nodes, such as edge creation, deletion, and attribute changes. Then the set of nodes $V(T) = \{i : \exists v_i(t) \in \mathcal{G}, t \in T\}$ and the set of edges $E(T) = \{(i, j) : \exists e_{ij}(t) \in \mathcal{G}, t \in T\}$ can be defined according to the above, while $\varphi_i^h(T) = \{j : (i, j) \in E(T), h \in \mathrm{Z}\}$ denotes the set of *h-hop* neighbours of node i in time interval $T = [0, t]$ and Z denotes the set of positive integers.

Recommended questions Given a set of users $U = \{u_1, u_2, \ldots, u_M\}$ and a set of items $K = \{k_1, k_2, \ldots, k_N\}$, where M denotes the number of users and N denotes the number of items. The items can be videos, products, information, etc. The user-item potential interaction matrix is $Y \in R^{M \times N}$ when $y_{uk} = 1$ indicates that the user interacts with items like clicking, favoriting, buying, etc., while $y_{uk} = 0$ is the opposite. In this paper, given the user-item interaction matrix Y and DKG $\mathcal{G}$, our goal is to train the function $\hat{y}_{uk} = \mathcal{F}(u, v | \Theta, Y, \mathcal{G})$ to predict the potential interest of user u in non-interactive item k, where Θ represents the parameters of the prediction function, and $\hat{y}_{uk}$ represents the probability of the user u interacting with item k. Table 1 summarizes the key notations used in this paper for ease of reference.

4 Proposed method

In this section, we first summarize the overall framework of the MTRDKG model. Next, we detail the design of the Information Sharing Unit (ISU), Recommendation Unit, and DKGE Unit. Finally, we discuss the training algorithm of the MTRDKG model.

Table 1 The key notations used in the paper

Symbol	Description
$\mathcal{G} = \{X(t) \mid t \in (t_1, t_2, \ldots, t_n)\}$	The set of time-stamped events
$v_i(t), e_{ij}(t)$	Node-wise event, interaction even
$T = [0, t], Z$	The time interval, the set of positive integers
$V(T) = \{i : \exists v_i(t) \in \mathcal{G}, t \in T\}$	The set of nodes
$E(T) = \{(i : j) : \exists e_{ij}(t) \in \mathcal{G}, t \in T\}$	The set of edges
$\varphi_i^h(T) = \{j : (i, j) \in E(T), h \in Z\}$	The set of h-hop neighbors of node i
$U = \{u_1, u_2, \ldots, u_M\}$	The set of users
$K = \{k_1, k_2, \ldots, k_N\}$	The set of items
$Y \in R^{M \times N}$	Interaction matrix
$\hat{y}_{uk} = \mathcal{F}(u, v \mid \Theta, Y, \mathcal{G})$	Goal function
$\rightarrow RS, \rightarrow DKGE$	The latent vectors of items and nodes
$M(\cdot), \sigma(\cdot)$	The fully connected network and non-linear activation function
ISU	The information sharing unit
$s_i(t)$	The memory state of node i at time point t
$s_i(t^-)$	The memory state of node i before time point t
$a(i, j)$	The attention coefficient
$m_i(t), M_i(t)$	The message and the aggregated message
ω	The hyperparameter controlling the time window
$h_i^{(l-1)}(t)$	The representation of node i at the $l-1$ th layer
$q^l(t), K^l(t), V^l(t)$	The query, keys, values
$\Phi(\cdot)$	The time encoding function
$H^{(l)}(t)$	The concatenated neighborhood representation
$\tilde{h}_i^{(l)}(t)$	The neighborhood representation calculated using attention mechanism
$z_i(t) = h_i^{(l)}(t)$	The temporal embedding of node i
$\mathcal{L}_{total}, \mathcal{L}_{RS}, \mathcal{L}_{DKGE}, \mathcal{L}_{REG}$	The loss function of overall, RS, DKGE, regularization term

4.1 Framework

This paper re-examines the relationships between nodes and edges in KGs and the relationships between KGs and items. Due to the strong correlation between items and nodes, the rich structured information and semantic associations in KGs can be transmitted to items through nodes, thereby influencing recommendation tasks. Simultaneously, due to this strong correlation, items will also affect nodes. However, SKGs lack dynamic information, leading to a series of issues such as timeliness. For example, Fig. 1 shows an example of a DKG in the YAGO dataset from time t_0 to t_n. From $t_0 \rightarrow t_1$,

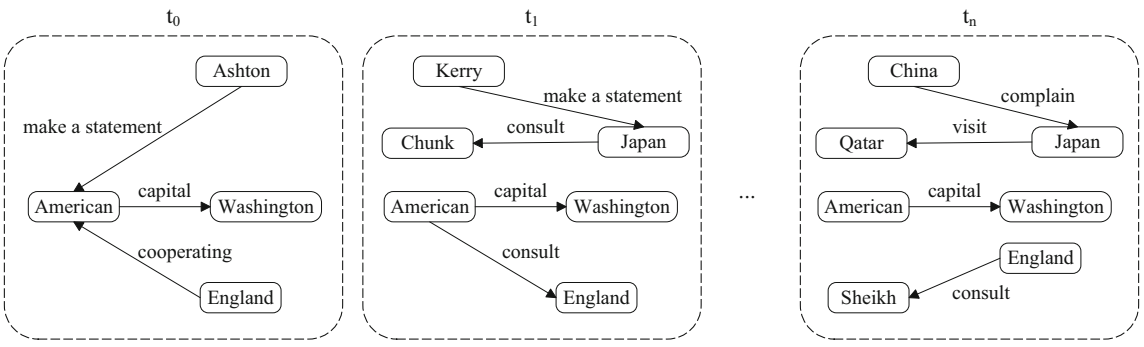


Fig. 1 A simple dynamic knowledge graph in the YAGO dataset

nodes like $Japan$, $Kerry$, and $Chunk$ were added, while the relationship between node $American$ and node $England$ changed. It is evident that nodes and their interrelationships in KGs constantly evolve, but these changes are not reflected in the KGs for SKGs. Based on this, we designed a multi-task knowledge-sharing strategy based on DKGs. This model models DKGs as continuous time-stamped events, updates attributes and relationships between nodes in real time, and cross-fuses node embeddings with recommendation items. Node embeddings contain feature information of domain nodes, temporal information, and edge feature information, effectively capturing the spatiotemporal structural features of DKGs. The overall framework of the MTRDKG is shown in Fig. 2.

4.2 Information-sharing units

In MTRDKG, we still adopt the cross-compression unit from MKR. However, unlike MKR, which learns high-order interaction features between items and entities, we use node embedding representations learned through TGN to replace entities. Given the item vector k and node embeddings $v = z_i(t)$, we use the L layer ISU to extract their feature expressions:

$$\begin{aligned} \left[v_{l+1}, k_{l+1}\right] &= ISU\left(v_l, k_l\right) \\ k_L &= ISU^L\left(v, k\right)[\mathrm{k}] \\ v_L &= ISU^L\left(v, k\right)[\mathrm{v}] \end{aligned} \tag{3}$$

Here, the suffixes [k] and [v] distinguish the output of the latent feature vectors of recommended items and nodes. At the l-th layer, we input the node embedding at time t and the corresponding item vector into the ISU for cross-compression operation, ultimately obtaining the item and node vectors for the l-th layer. The calculation expression for ISU is:

$$k_{l+1} = C_l \cdot w_k^1 + C_l^T \cdot w_k^2 + b_k \tag{4}$$

$$v_{l+1} = C_l \cdot w_v^1 + C_l^T \cdot w_v^2 + b_v \tag{5}$$

Where C_l represents the cross matrix obtained by the cross multiplication of the item vector and the node embedding vector, and C_l^T is its transpose matrix. Introducing learnable parameters such as w_k^1, w_v^1, w_k^2, w_v^2, b_k, and b_v to compress the high-dimensional matrix into a low-dimensional vector.

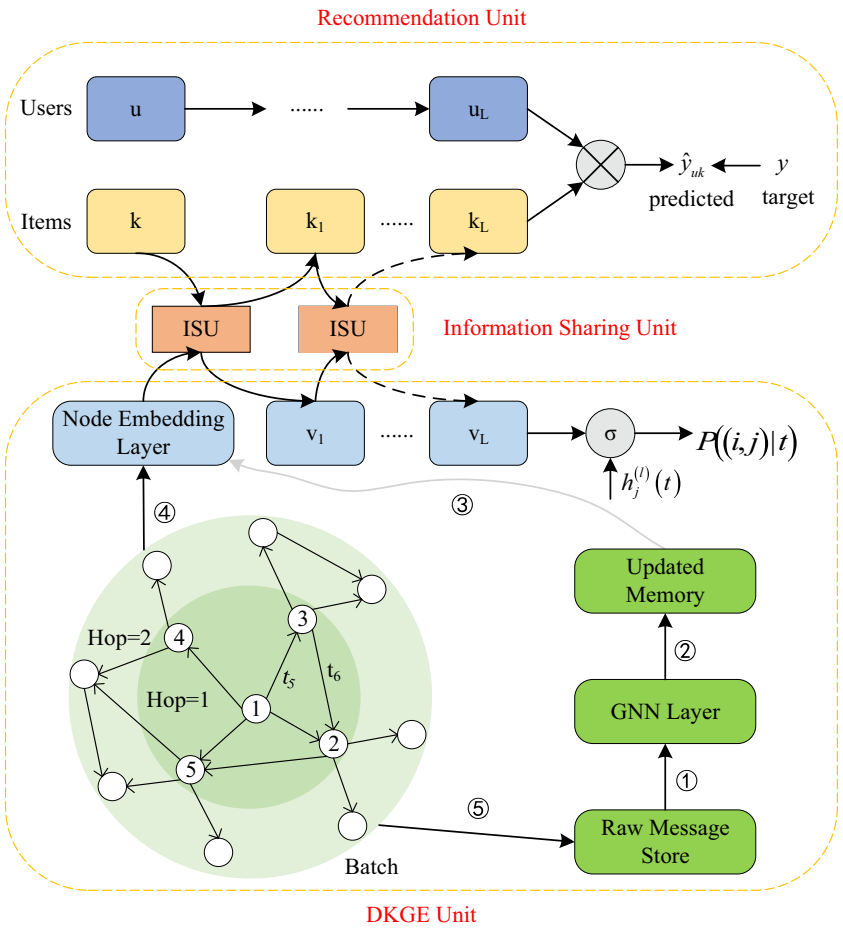


Fig. 2 MTRDKG model structure diagram. (a) Recommendation unit: Connects the user vector processed through multiple MLP layers with the item vector fused with node information to predict user interests. (b) ISU: Used to generate cross features of items and nodes and adaptively control information transfer. (c) DKGE unit: The core part is TGN. First, batch raw messages are passed to the GNN layer for message passing and aggregation to update node states (1, 2). Second, the updated node states are passed to the node embedding layer to compute node embeddings (3, 4), and the node representations fused with item information are obtained through the information sharing unit (ISU) for future link prediction. Lastly, interaction information during this batch process is stored (5)

4.3 RS unit

We utilize an L-layer multi-layer perceptron (MLP) to learn the user's latent low-dimensional dense features. Given the user feature vector u, the MLP operation expression is:

$$u_L = M(M(\ldots M(u))) = M^L(u) \tag{6}$$

$$M(u) = \sigma(Wu + b) \tag{7}$$

Where $M(\cdot)$ denotes the fully connected network, and $\sigma(\cdot)$ represents the non-linear activation function. W and b represent the weight matrix and bias, respectively. We then use the L-layer ISU to fuse the items with the node embeddings to introduce valid side information for the RS:

$$k_L = ISU^L(v, k)\,[\mathrm{k}] \tag{8}$$

With the calculation of (3-8), we have obtained the user and item embedding expressions for layer L. Next, the expression for the user preference prediction function for the item is:

$$\hat{y}_{uk} = f_{RS}(\mathrm{u_L}, \mathrm{k_L}) \tag{9}$$

Where $\hat{y}_{uk}$ denotes the probability value of user u interacting with item k.

4.4 DKGE module

Reference [16], the MTRDKG model views each node's representation as a time series, generating graph nodes $Z(t) = \left(z_1(t), z_2(t), \ldots, z_{n(t)}(t)\right)$ at any given time t. Below, the core components of the MTRDKG model and the complete DKGE are introduced.

4.4.1 Core components

Memory As node representations in DKG are constantly changing, tracking and storing the historical information of nodes is necessary. The memory at time t consists of the vectors $s_i(t)$ for all nodes i. When an event related to node i occurs, the memory for that node is updated. For new nodes that have not appeared before, a corresponding zero vector is set up in the memory, and the information for the new nodes is updated after events involving the new nodes occur.

Raw message store For time t, this module stores the raw information of the most recent interaction before t for each node i, which will serve as the input for the GNN layer.

GNN layer GNN Layer. At the time t, assume the existence of a source node i and a target node j, along with their interaction event $e_{ij}(t)$. The role of the GNN layer is to aggregate the messages iteratively passed from the target node j to the source node i and update the memory of node i through bidirectional information flow. Reference [16] uses a simple concatenation operation for message passing, making it difficult for the model to differentiate the importance of different nodes and edges. Inspired by the development of graph convolutional networks (GCN) [39], we designed a novel recursive embedding propagation method based on the GCN architecture. This method utilizes graph attention networks, allowing for different levels of importance to be assigned to nodes within the neighborhood of each node and generating attention weights for cascading embedding propagation. Executing this recursive embedding mechanism allows the model to learn complex interaction dependencies between nodes with linear time complexity. The GNN layer includes two parts: message passing and message aggregation.

The input for the message passing part includes $s_i(t^-)$, $s_j(t^-)$, and $e_{ij}(t)$, representing the vectors stored in memory for nodes i and j before time t (starting from the time of the last event involving the nodes, with Δt representing the time difference since the event occurred), as well as the interaction between the nodes. Therefore, (1) can be expanded as:

$$m_i(t) = \sum_{j \in \varphi_i^b(t)} a(i,j)\, Concat\left(s_i\left(t^-\right), s_j\left(t^-\right), e_{ij}(t), \Delta t\right) \tag{10}$$

Where $a(i, j)$ represents the attention coefficient, used to indicate the importance of the target node j to the source node i. $Concat(\cdot)$ denotes the concatenation operation.

Notably, when a node-level event $v_i(t)$ occurs, the message passing only involves the node itself, with $a(i, j) = 1$:

$$m_i(t) = Concat(s_i(t^-), v_i(t), \Delta t) \tag{11}$$

TGN adopts a batch processing mechanism for computational efficiency, however, multiple events involving node i may occur at different times. After each event occurs and generates a message, it is necessary to aggregate the various messages (such as $m_i(t_1), \ldots, m_i(t_c)$, where $t_1, \ldots, t_c \le t$). To avoid aggregating outdated and irrelevant messages, we have carefully designed an aggregation function, with the main steps as follows: (1) Set a time window: an adjustable hyperparameter ω. (2) Filter messages: traverse all messages related to node i and their corresponding times, only retaining messages within the time window, which we refer to as effective messages. (3) Apply the aggregation function: use the average aggregation method.

$$M_i(t) = \frac{1}{|I|} \sum_{t_c \in T'} m_i(t_c) \tag{12}$$

where $T' = \{t_c | t - t_c \leq \omega, \forall t_1, \ldots, t_c\}$ denotes the set of time of valid messages. I is the number of valid messages. We set the ω value to 1 (in seconds), indicating that only the most recent messages are retained. This is because it has been shown that the most recent edges often contain the maximum amount of information.

Memory update module After an event involving a node occurs, the node's memory is updated using the gated recurrent unit (GRU):

$$s_i(t) = GRU(M_i(t), s_i(t^-)) \tag{13}$$

Note that (13) only illustrates the node-level events. When interaction events are involved, it is necessary to update the memory of the interacting nodes separately.

Node embedding layer To prevent the memory information of node i from becoming outdated, the attributes of nodes and edges, along with historical information at different time steps, are encoded into continuous vectors to generate the embedding representation $z_i(t)$ of node i at any time t:

$$z_i(t) = \sum_{j \in \varphi_i^h([0,t])} h(s_i(t), s_j(t), e_{ij}, v_i(t), v_j(t))$$

Where $h(\cdot)$ is a learnable function. Specifically, different node embedding functions will produce different approaches. According to the scenarios listed in [16], we obtain the following variants:

MTRDKG (dir): $z_i(t) = s_i(t)$. Direct embedding method, which directly defines the embedding of node i at time t as the node state.

MTRDKG (tf): $z_i(t) = (1 + w\Delta t) \odot s_i(t)$. The time projection method defines the node embedding at time t as a nonlinear combination of the node's state and the time difference. Here, $\odot$ denotes element-wise vector multiplication, and w is a learnable parameter.

MTRDKG (sum): Summation method. Calculates the embedding of a node at time t through the weighted sum of the node's historical states:

$$\tilde{H}_i^{(l)}(t) = RELU\Big(\sum_{j \in \varphi_i^h([0,t])} W_1^{(l)} Concat(h_j^{(l-1)}(t), e_{ij}, \Phi(t - t_j))\Big) \tag{14}$$

$$h_i^{(l)}(t) = W_2^{(l)} Concat\left(h_i^{(l-1)}(t), \tilde{H}_i^{(l)}(t)\right) \tag{15}$$

Where $h_i^{(l-1)}(t)$ represents the representation of node i at time t at the $l-1$-th layer. And the initial node representation is $h_i^{(0)}(t) = s_i(t) + v_i(t)$. $\Phi(\cdot)$ denotes the time encoding function using the Time2Vec method, which is also used in [16, 27, 40]. $\tilde{H}_i^{(l)}(t)$ represents aggregated information, and $z_i(t) = h_i^{(l)}(t)$.

MTRDKG Attention mechanism method. It takes the neighborhood representation $H = \left\{h_1^{(l-1)}(t_1), \ldots, h_N^{(l-1)}(t_N)\right\}$ of node i at time t and the corresponding interaction features $\{e_{i1}(t_1), \ldots, e_{iN}(t_N)\}$ as inputs for TGN:

$$H^{(l)}(t) = Concat(h_1^{(l-1)}(t), e_{i1}(t_1), \Phi(\Delta t_1), \ldots, h_N^{(l-1)}(t), e_{iN}(t_N), \Phi(\Delta t_N)) \tag{16}$$

$$q^{(l)}(t) = Concat\left(H^{(l)}(t), \Phi(0)\right) \tag{17}$$

$$K^{(l)}(t) = V^{(l)}(t) = H^{(l)}(t) \tag{18}$$

$$\tilde{h}_i^{(l)}(t) = MultiHeadAtte^{(l)}\left(q^{(l)}(t), K^{(l)}(t), V^{(l)}(t)\right) \tag{19}$$

$$z_i(t) = h_i^{(l)}(t) = MLP^{(l)}\left(Concat\left(h_i^{(l-1)}(t), \tilde{h}_i^{(l)}(t)\right)\right) \tag{20}$$

Where $H^{(l)}(t)$ represents the concatenated neighborhood representations. $\Delta t_N = t - t_N$ denotes the time difference of interactions. At each layer, we use a multi-head attention mechanism to compute the neighborhood aggregation information $\tilde{h}_i^{(l)}(t)$, where the query $q^{(l)}(t)$ represents the target node, and the keys $K^{(l)}(t)$ and values $V^{(l)}(t)$ respectively represent the neighborhood nodes of the target node. Finally, an MLP is used to compute the aggregated information and the target node representation to obtain the target node embedding. This paper defaults to using the attention mechanism method.

4.4.2 Complete DKGE

The node embedding representation $z_i(t)$ aggregating neighborhood information of nodes is obtained through (16-20). Unlike MKR, which cascades the head entities and relations before predicting the tail entity, MTRDKG inputs $z_i(t)$ into

L layers of ISU to obtain the node vector representation v_L and then uses the $Sigmoid$ function $\sigma(\cdot)$ for future link prediction:

$$v_L = ISU^L(v,k)[\mathrm{v}] \tag{21}$$

$$P((i,j)|t) = \sigma\left(v_L W h_j^{(l)}(t)\right) \tag{22}$$

Where W is the weight matrix of the fully connected layer, and $P((i,j)|t)$ denotes the probability of an edge occurring at two nodes at a given time t.

4.5 Training algorithms

The overall loss function $\mathcal{L}_{total}$ for the MTRDKG method:

$$\begin{aligned} \mathcal{L}_{total} &= \mathcal{L}_{RS} + \mathcal{L}_{DKGE} + \mathcal{L}_{REG} \\ \mathcal{L}_{RS} &= \sum_{u\in U, k\in K} \mathcal{J}\left(\hat{y}_{uk}, y_{uk}\right) \\ \mathcal{L}_{DKGE} &= \sum_{(v_i,v_j,t_{ij})\in E(T)} -\log\left(\sigma\left(-vh_j^{(l)}(t_{ij})\right)\right) - QE_{v_q\sim P_n(v)}\log\left(\sigma\left(vh_q^{(l)}(t_{ij})\right)\right) \\ \mathcal{L}_{REG} &= \lambda\|\Theta\|_2^2 \end{aligned} \tag{23}$$

Algorithm 1 Multi-Task Alternately Training for MTRDKG.

Input: Y - User-Item interaction matrix in the recommendation task;
$s \leftarrow 0$ - initialise memory to zeros;
$m\_raw \leftarrow \{\}$ - initialise the raw messages;
Output: $z_i(t)$ - node temporal embedding;
$s_i(t)$ - the memory of a node;
$\mathcal{F}(u, v|\Theta, Y, \mathcal{G})$ - prediction function;

1: Initialize parameters
2: **for** r=1 to epo **do** *// epo is the number of training times*
// train RS task
3: **for** t=1 to tim **do** *// tim is the number of training iterations*
4: Sample iterations from Y;
5: Update parameters of in (3-4), (6-9), (23);
6: **end for**
// train DKGE task
7: **for** batch (i, j, e_{ij}, t) **do** *//Sampling a batch from the training data*
8: Update parameters of $\mathcal{F}(\cdot)$ in (3), (5), (10)-(23);
9: **end for**
10: **end for**

In (23), $\mathcal{L}_{RS}$ represents the loss function for the recommendation task, calculated by the cross-entropy loss function $\mathcal{J}$. $\mathcal{L}_{DKGE}$ represents the loss function for the DKGE task. Unlike MKR, which calculates the difference in scores of true and false triples to represent the KGE task loss function, we use the logarithmic loss function to calculate the inner product of the representation vectors between observed edges (interaction between nodes v_i and v_j at time t_{ij} and unobserved edges (extracting the number of Q nodes from the negative sample distribution space $P_n(v)$ and then summing them up. This encourages the model to learn a more effective node representation. $\mathcal{L}_{REG}$ is the L2 regularization term used to prevent overfitting, and λ is its weight. We train MTRDKG using an alternating training strategy, as detailed in Algorithm 1. A training epoch is denoted by epo, with each epoch consisting of two phases: the first phase trains the RS task (lines 3-7), and the second phase trains the DKGE task (lines 8-9). We set a hyperparameter tim to control the frequency of alternating training of the DKGE task [12]. This means that after training the RS recommendation tim times, the DKGE task is trained once. As our goal is to enhance the performance of the RS task, the DKGE task serves only as a regularization constraint for the RS task.

5 Experiments

In this chapter, we will evaluate the performance of the proposed MTRDKG model from the following five aspects:

- RQ1: Does the MTRDKG model perform better than the state-of-the-art recommendation methods and DKGE methods?
- RQ2: Do different embedding methods significantly affect the MTRDKG model?
- RQ3: Is the training efficiency of the MTRDKG model superior to other methods?
- RQ4: Does multi-task joint training lead to poorer convergence of the model?
- RQ5: Do model parameters significantly affect the MTRDKG model?

5.1 Datasets and data preprocessing

To verify the effectiveness of MTRDKG, we conduct experiments on four public datasets: social, encyclopedia, movie,

and music. The descriptions of the four datasets are as follows:

- **Reddit dataset**[1]. It contains many Reddit posts and comments. We use the most active users and subreddit as graph nodes, while users submitting posts or posting comments are considered interactions.
- **Wikipedia dataset**[2]. It is a dataset with many Wikipedia articles and edit histories containing information about user interactions over 30 days. We use users and pages as graph nodes, with a user editing or creating a page represented as an interaction.
- **MovieLens-1M dataset**[3]. It is a widely used recommender system dataset containing 1000209 rating records from 6000 users for 4000 movies with a rating range of 1-5. We represent users and movies as nodes, and edges represent interactions between users and movies, i.e. ratings.
- **Last.FM dataset**[4]. Records of the listening history of users of the online music platform Last.FM over one month. We selected 1000 users and 1000 most popular music tracks and represented users and music as graph nodes, with edges representing users' listening records to music.

The Reddit and Wikipedia datasets are bipartite interaction graphs, where we convert textual features such as users' posts and editing content into 172-dimensional vector representations under LIWC categories [41]. Wang et al. [12] set a threshold of 4 to explicitly convert the explicit feedback data in the MovieLens-1M dataset into positive samples (i.e., ratings above the threshold, marked as 1) and negative samples (ratings below the threshold, marked as 0). Last.FM and MovieLens-1M do not use LIWC; therefore, we use Word2Vec to convert them into 128-dimensional vector representations. To adapt to the recommendation task, we treat the source nodes in KG as the Users set, while the target nodes are the Items set. Our experimental setup is similar to [27], dividing the dataset into a training set (70%) - validation set (15%) - test set (15%), but the difference is that we focus on recommendation performance. More statistical information can be seen in Table 2.

5.2 Comparable methods

To validate the recommendation performance and DKGE capability of MTRDKG, we compare it with the following state-of-the-art methods: recommendation methods (Wide & Deep, RippleNet, KGCN, MKR, KGARA, EMKR) and DKGE methods (CTDNE, TGAT, TGNs, AdaNet, DGSR).

- **Recommended methods**: Wide & Deep [42] is a deep learning model that combines linear models and deep neural networks to address RS's memorization and generalization issues. RippleNet [43] simulates the multi-hop reasoning process of users in KGs, automatically learning the latent representations of items and users for efficient and interpretable personalized recommendations. KGCN [3] leverages GCNs to capture the structural information of entities and relationships in KGs. MKR [12] facilitates mutual enhancement between knowledge graph-enhanced recommendation tasks and entity-linking tasks through shared representation learning and cross-compression units. KGARA [44] combines semantic information in KGs with user-item interaction data, learning latent representations of users and items through GNNs and an adaptive relational attention mechanism. EMKR [36] utilizes attention mechanisms to discover historical information of MKR users and employs relation-aware GCNs to mine deep relational domain features of entities in KGs.
- **DKGE Methods**: CTDNE [23] is a continuous-time dynamic network embedding method designed to learn vector representations of nodes in dynamic networks. TGAT [27] is an inductive representation learning model on temporal graphs, aiming to capture dynamic graphs' structural and temporal characteristics. TGNs [16] is a deep learning model that learns representations of nodes in dynamic graphs by integrating KG and temporal information. The core of AdaNet [45] is a knowledge adaptation module, which adapts and integrates knowledge learned across different time windows. DGSR [46] uses dynamic graph neural networks as the base model to extract user preferences from DKG.

5.3 Parameter settings

The MTRDKG model was implemented using PyCharm as the IDE platform. The operating system used was Windows 10 Home Chinese Edition, and the programming language utilized was Python 3.8. The CPU employed was i7-7700HQ, GeForce RTX 2080 Ti, and the PyTorch version utilized was 1.4. In MTRDKG, the recommended score function f_{RS} is the inner product, and $\lambda = 10e - 6$ denotes the regularisation weights. Other hyperparameters are set as shown in Table 3. Where L is the number of ISU layers used to extract shared information, tim is the training frequency, epo is the training epoch, ks and $batch$ are the batch sizes of RS and DKGE, respectively, and lr_{rs} and lr_{dkge} are the learning

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

[2] http://snap.stanford.edu/jodie/wikipedia.csv

[3] https://grouplens.org/datasets/movielens/1m/

[4] https://grouplens.org/datasets/hetrec-2011/

Table 2 Statistical information on the data set used for the experiment

	Reddit	Wikipedia	Last.FM	MovieLens-1M
# Users	10000	8227	1000	6000
# Items	1000	1000	1000	4000
# Nodes	11000	9227	2000	10000
# Edges	672447	157474	1293103	1000209
# Feature dimension	172	172	128	128
# Feature type	LIWC category	LIWC category	Word2Vec	Word2Vec
Timespan	30 days	30 days	30 days	34 months
Chronological Split	70%-15%-15%	70%-15%-15%	70%-15%-15%	70%-15%-15%

rates. *md* denotes the memory dimension, and *nd* means the user, item, node embedding, and temporal embedding dimensions. *heads* represents the number of multi-headed attention mechanism heads, and *dr* indicates the dropout rate. *h* represents the number of TGN layers (hops). To find the most appropriate parameters, L is taken from {1, 2, 3, 4, 5}, tim from {1, 2, 3, 4, 5, 6}, *heads* from {1, 2, 3, 4, 5} and h from {1, 2, 3}. We conducted comparison experiments in three scenarios: (1) CTR prediction: using *AUC* and *ACC* as evaluation metrics. (2) Top-K recommendation: using *Precision@K* and *Recall@K* as evaluation metrics. (3) Future link prediction: using *AUC* and *Average Precision* (*AP*) as evaluation metrics. Where (1) and (2) are used to assess recommendation performance, while (3) uses unseen nodes to derive future links to measure the capability of DKGE. We will run each dataset 10 times to reduce random errors, and the final results will be averaged.

5.4 Performance comparison (RQ1)

5.4.1 Recommendation performance

To explore the recommendation performance of our model, this section conducts comparative experiments under CTR prediction and Top-K recommendation. Based on the experimental results, we can draw the following conclusions:

From Table 4, it can be observed that Wide & Deep and RippleNet perform poorly on the Last.FM dataset but show better performance on MovieLens-1M. This indicates that these two methods struggle to capture user preferences when the data is sparse. This is because Last.FM is sparser, with a data density of only 0.59%, whereas MovieLens-1M is denser, with a data density of 5.32%. KGCN performs well in both sparse and dense scenarios. This may be because KGCN incorporates GNN, enabling it to explore user interests by mining multi-hop neighborhood information. KGARA and KGCN have comparable performance in MovieLens-1M, but KGARA performs worse in Last.FM. EMKR ranks second on both AUC and ACC metrics in the Last.FM and MovieLens-1M datasets, likely because EMKR uses attention mechanisms and relation-aware GCN to mine the structured information of KG fully. MTRDKG performs the best across all metrics in the two datasets. Specifically, compared to MKR, the performance of MTRDKG increases by between 1.6% and 2.9%, indicating that MTRDKG is an effective enhancement of MKR. In MovieLens-1M, MTRDKG improves the AUC and ACC values by 1.3% and 0.9%, respectively, compared to the best-performing baseline method. In Last.FM, the AUC, and ACC values of MTRDKG increase by 0.5% and 1.5%, respectively. This is because, compared to these static methods, MTRDKG uses dynamic techniques such as time aggregation, information propagation, and embedding updates, allowing for better representation learning in sparse data scenarios. The standard deviation of MTRDKG is also the smallest, indicating the method's stable performance.

Compared to the CTR prediction experiments, we also conducted comparative experiments in the Top-K recommendation scenario, introducing two additional datasets, Reddit and Wikipedia. From Figs. 3 and 4 shows that under the Precision@K metric, MTRDKG significantly outperforms the other baseline methods. In Recall@K, MTRDKG performs the best in most cases, on the Last.FM dataset, the performance is not the best when the K value is small. This might

Table 3 Parameter setting status

DATASETS	L	tim	epo	ks	$batch$	lr_{rs}	lr_{dkge}	md	nd	$heads$	dr	h
Reddit	3	5	15	2048	200	0.02	0.001	172	64	2	0.1	1
Wikipedia	3	5	15	4096	200	0.02	0.001	172	64	2	0.1	1
MovieLens-1M	3	2	20	256	200	0.001	0.0001	128	32	2	0.1	1
Last.FM	2	3	10	256	200	0.001	0.0001	128	16	2	0.1	1

Table 4 Experimental results of MTRDKG and baseline methods in CTR prediction

APPROACHES		MovieLens-1M		Last.FM	
		AUC	*ACC*	*AUC*	*ACC*
Wide & Deep	*Ave*	0.898(−4.0%)	0.820(−4.3%)	0.756(−7.0%)	0.688(−10%)
	Std	/	/	/	/
RippleNet	*Ave*	0.916(−2.1%)	0.842(−1.7%)	0.748(−7.9%)	0.713(−6.7%)
	Std	0.001	0.001	0.007	0.006
KGCN	*Ave*	0.901(−3.7%)	0.824(−3.8%)	0.801(−1.4%)	0.729(−4.6%)
	Std	0.002	0.002	0.005	0.004
MKR	*Ave*	0.917(−2.0%)	0.843(−1.6%)	0.791(−2.6%)	0.742(−2.9%)
	Std	0.001	0.002	0.009	0.013
KGARA	*Ave*	0.901(−3.7%)	0.827(−3.4%)	0.791(−2.6%)	0.715(−6.5%)
	Std	0.001	0.001	0.005	0.006
EMKR	*Ave*	0.923(−1.3%)	0.848(−0.9%)	0.808(−0.5%)	0.753(−1.5%)
	Std	0.001	0.001	0.002	0.002
MTRDKG	*Ave*	**0.935**	**0.856**	**0.812**	**0.764**
	Std	0.001	0.001	0.001	0.002

Note: (+/-) indicates the degree of lift or drop relative to MTRDKG, with the best value in bold. *Ave* and *Std* represent the average value and standard deviation

be because Last.FM is relatively sparse, making the dynamic features less pronounced than static features, and some static models perform better at smaller K values due to their simplified assumptions. When the K value increases, MTRDKG performs the best, as introducing the time dimension provides an advantage in capturing dynamic long-term dependencies. Compared to MKR, MTRDKG performs better under any condition, proving that our proposed method offers significant improvements.

5.4.2 DKGE performance

To explore the impact of information sharing within the MTL framework on real-time updates of DKG, we conducted a comparative future link prediction experiment. The experimental results are shown in Table 5. CTDNE performs poorly on Reddit and Wikipedia. For instance, the AUC value decreases by 58.4% in the Reddit dataset, while the AP value decreases by 48.4%. In Wikipedia, the AUC and AP values decrease by 41.6% and 36.2%, respectively. TGAT performs better on the Reddit dataset, possibly because Reddit's dynamic data helps capture the time-varying nature of entities and relationships. TGNs, AdaNet, and DGSR achieve excellent results. Specifically, AdaNet ranks first in the AP metric on Reddit, with all other results being second. Meanwhile, TGNs and DGSR take third place in all metrics. MTRDKG consistently outperforms other methods except for the AP metric on Reddit, indicating that introducing item knowledge is beneficial for the adaptive update of DKG.

5.5 Model analysis

5.5.1 Analysis of different MTRDKG variants (RQ2)

To study the impact of different node embedding methods on the model's recommendation performance, we compare MTRDKG with three variants. The variants are represented in abbreviated form as follows.

From Table 6 and Fig. 5, we can draw the following conclusions: (1) The *tf* method performs worse than the *dir* method. This is because the *tf* method captures temporal information through simple non-linear combinations, unable to effectively handle complex structures and temporal changes in DKG. Although the *dir* method does not explicitly consider temporal information, it relies on other parts of the model (such as message passing and memory updates) to implicitly capture temporal information. (2) The performance of *tf* and *dir* methods is inferior to the *sum* method and MTRDKG. This is because the *sum* method and MTRDKG can access deeper and more critical semantic information, and they generate embeddings by calculating the historical information of nodes and information of neighboring nodes, better reflecting the structural and temporal information in DKG. (3) MTRDKG performs better than the *sum* method. MTRDKG uses a graph attention mechanism to calculate node embeddings, which allows for adjusting weights based on the importance of nodes and the state of neighboring nodes, offering greater flexibility. In contrast, the *sum* method calculates node embeddings simply by a weighted *sum* of

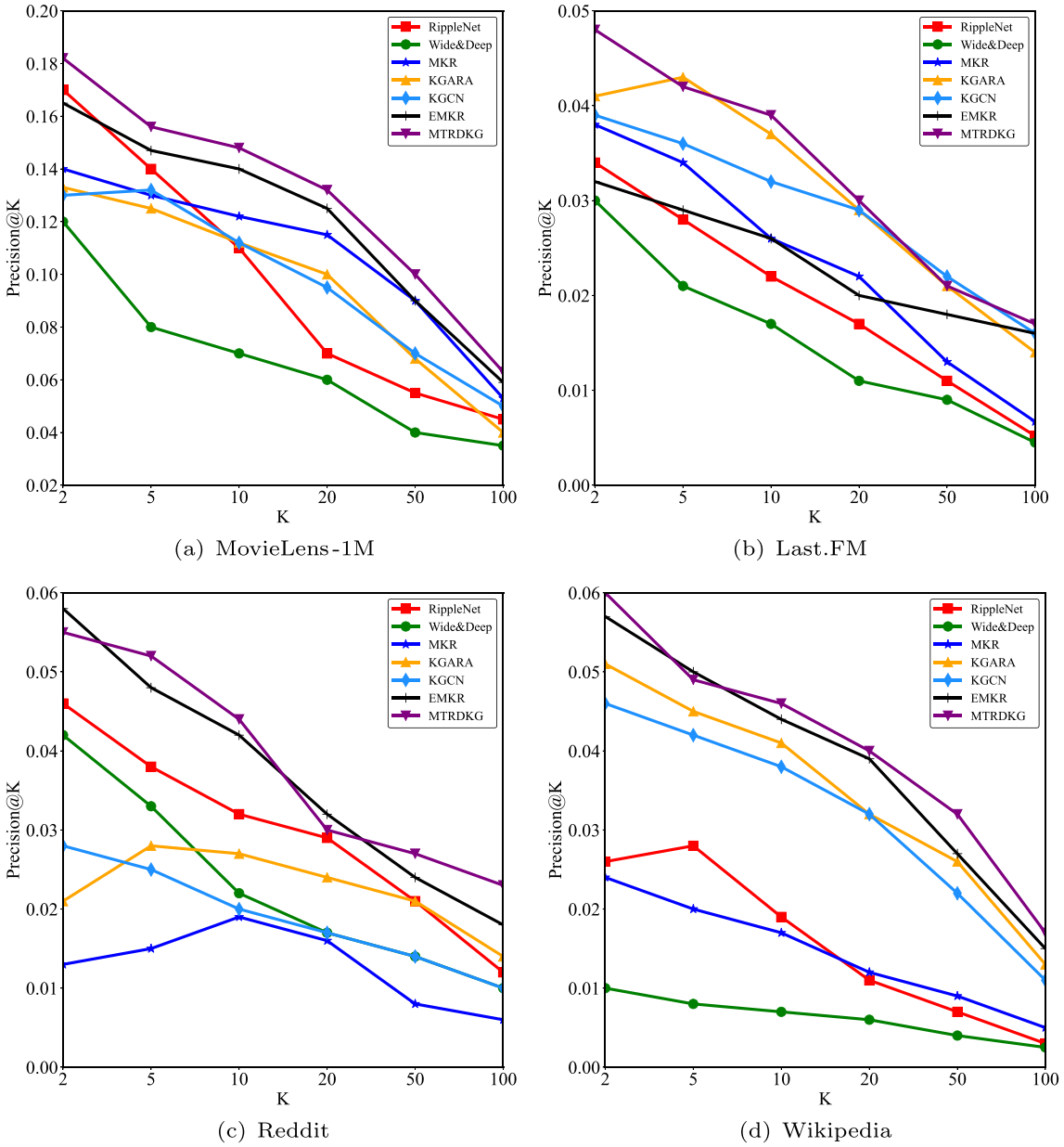


Fig. 3 MTRDKG vs baseline methods for Precison@K results in Top-K recommendations

the node's historical states. However, the only disadvantage of MTRDKG is that it trains slightly slower than the *sum* method. Overall, MTRDKG, based on the attention mechanism, performs the best.

5.5.2 Efficiency comparison(RQ3)

To validate the efficiency of MTRDKG, we compared training times with the five best-performing baseline methods on the Last.FM and MovieLens-20M datasets. The purpose of using MovieLens-20M is to assess the model's processing efficiency for large-scale datasets. The results are shown in Fig. 6. Wide & Deep has the shortest training time, likely due to its simpler structure. Compared to multitask models like MKR, EMKR, and MTRDKG, single-task methods like RippleNet and KGCN have longer training times. This indicates that parameter sharing in the MTL framework can enhance model efficiency. In Last.FM, MTRDKG has a longer training time than MKR but a shorter runtime on the large-scale dataset MovieLens-20M. Although MTRDKG is not the

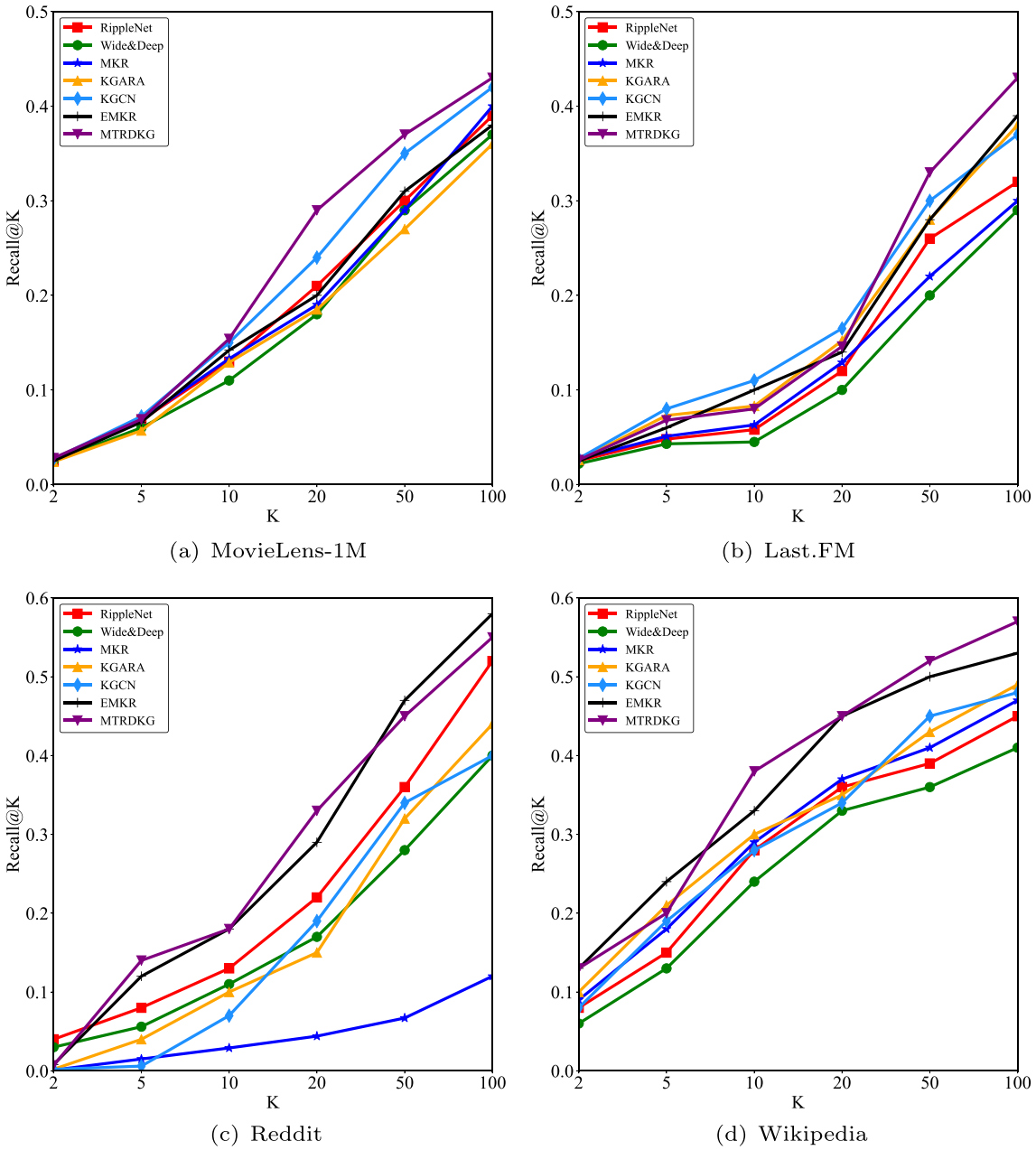


Fig. 4 MTRDKG vs baseline methods for Recall@K results in Top-K recommendations

Table 5 Experimental Results of MTRDKG and baseline methods in future link prediction

APPROACHES	Reddit		Wikipedia	
	AUC	*AP*	*AUC*	*AP*
CTDNE	0.414(−58.4%)	0.512(−48.4%)	0.579(−41.6%)	0.635(−36.2%)
TGAT	0.975(−2.1%)	0.970(−2.3%)	0.950(−4.2%)	0.947(−4.7%)
TGNs	0.986(−1.0%)	0.987(−0.6%)	0.984(−0.8%)	0.985(−1.0%)
AdaNet	0.993(−0.3%)	**0.994**(+0.2%)	0.988(−0.4%)	0.989(−0.6%)
DGSR	0.981(−1.5%)	0.977(−3.4%)	0.981(−1.1%)	0.986(−0.8%)
MTRDKG	**0.995**	0.992	**0.991**	**0.994**

Table 6 Results of AUC and ACC comparison between MTRDKG and the three variants in CTR prediction

Models	MovieLens-1M		Last.FM		Reddit		Wikipedia	
	AUC	*ACC*	*AUC*	*ACC*	*AUC*	*ACC*	*AUC*	*ACC*
MTRDKG(dir)	0.933	0.853	0.809	0.759	0.985	0.991	0.987	**0.994**
	(−0.2%)	(−0.3%)	(−0.3%)	(−0.6%)	(−1.0%)	(−0.1%)	(−0.4%)	
MTRDKG(tf)	0.931	0.850	0.809	0.756	0.983	0.985	0.984	0.989
	(−0.4%)	(−0.7%)	(−0.3%)	(−1.0%)	(−1.2%)	(−0.7%)	(−0.7%)	(−0.5%)
MTRDKG(sum)	**0.935**	0.853	0.811	**0.766**	0.994	0.990	0.989	**0.994**
		(−0.3%)	(−0.1%)	(+0.3%)	(−0.1%)	(−0.2%)	(−0.2%)	
MTRDKG	**0.935**	**0.856**	**0.812**	0.764	**0.995**	**0.992**	**0.991**	**0.994**

method with the shortest training time, its performance is the best, making these additional training times acceptable.

5.5.3 Convergence analysis(RQ4)

In this section, we increase the training epochs to extend the training time on the MovieLens-1M dataset to verify the trend of MTRDKG's accuracy over time. Figure 7 shows the evolution patterns of AUC and ACC under different training rounds, where (a) and (b) correspond to the training dynamics when using AUC and ACC as test metrics, respectively. We can observe that MTRDKG converges very quickly. Specifically, the MTRDKG model begins to converge after about 15 training rounds. The model's performance then fluctuates within a minimal range, demonstrating strong stability.

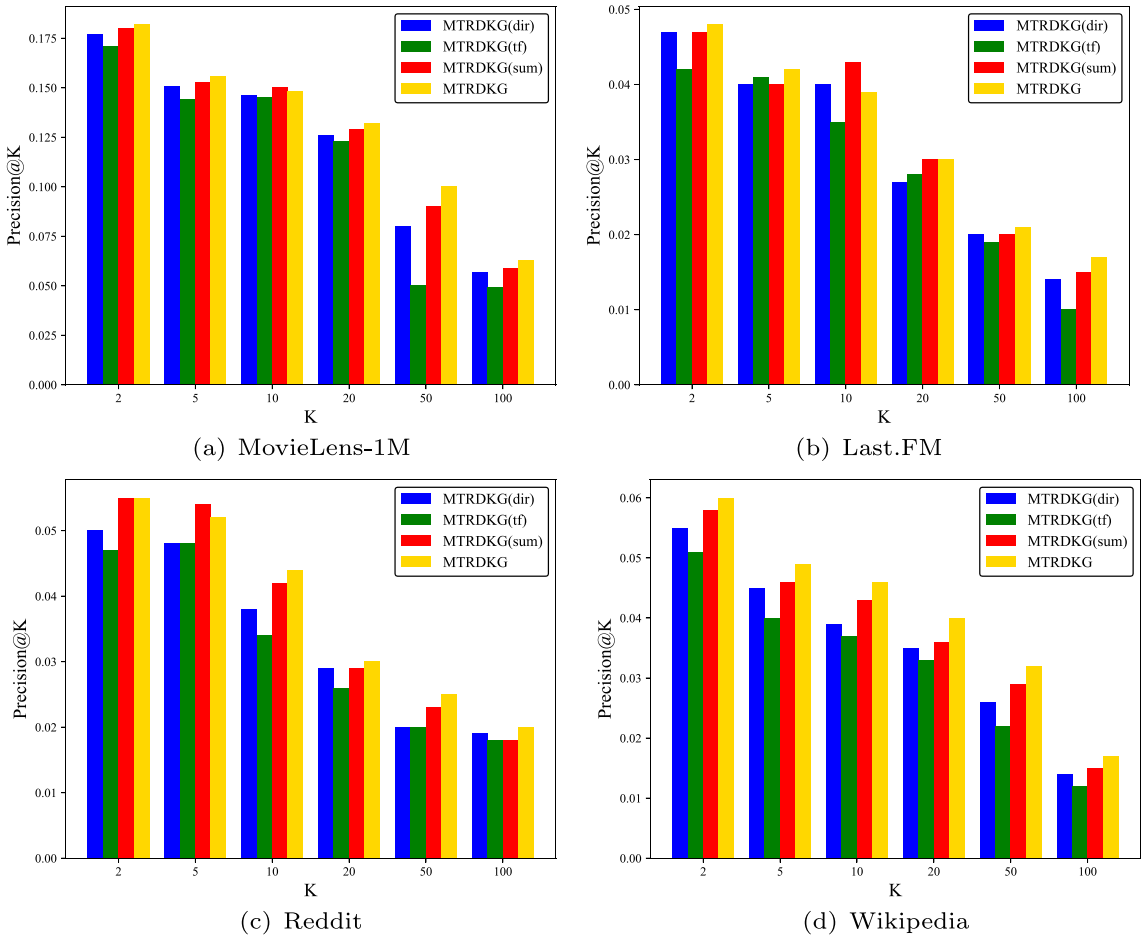
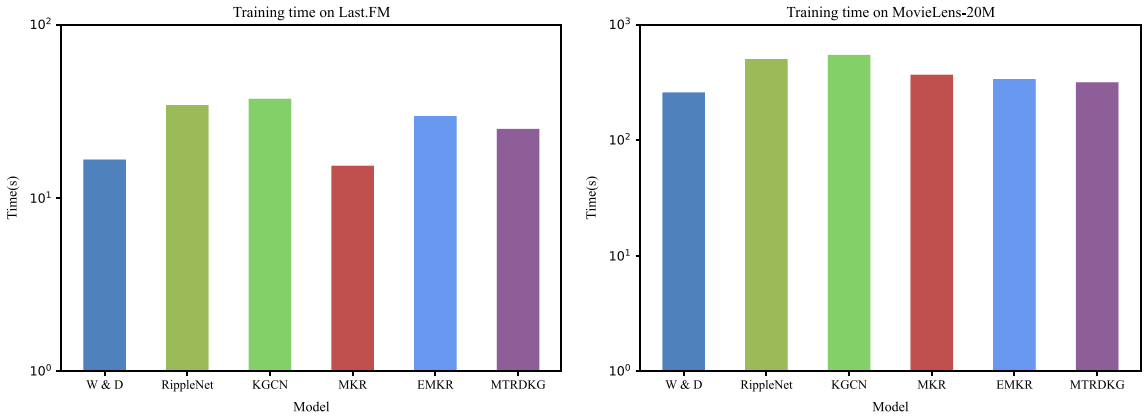


Fig. 5 Comparison results of MTRDKG with three variants of Precison@K in Top-K recommendations



(a) Last.FM (b) MovieLens-20M

Fig. 6 The training time of different models in the Last.FM and MovieLens-20M datasets

5.6 Parameter sensitive analysis(RQ5)

5.6.1 Impact of embedded dimensions

We increased the dimension of *nd* from 4 to 128 to explore its impact on the MTRDKG model. Figure 8 (a) and (b) show that on the MovieLens-1M dataset, the model's performance steadily improves as the dimension of *nd* increases. Still, after exceeding a dimension of 32, the *ACC* value decreases. Although the *AUC* value continues to rise, the increase is minimal. To achieve a balance between the two metrics, we set the dimension of *nd* to 32. On the Last.FM dataset, the best performance is observed when the dimension of *nd* is 16, with more significant fluctuations in performance upon further increases. This may be because increasing the dimension can initially encapsulate more helpful information, but continued increases incorporate more noise data.

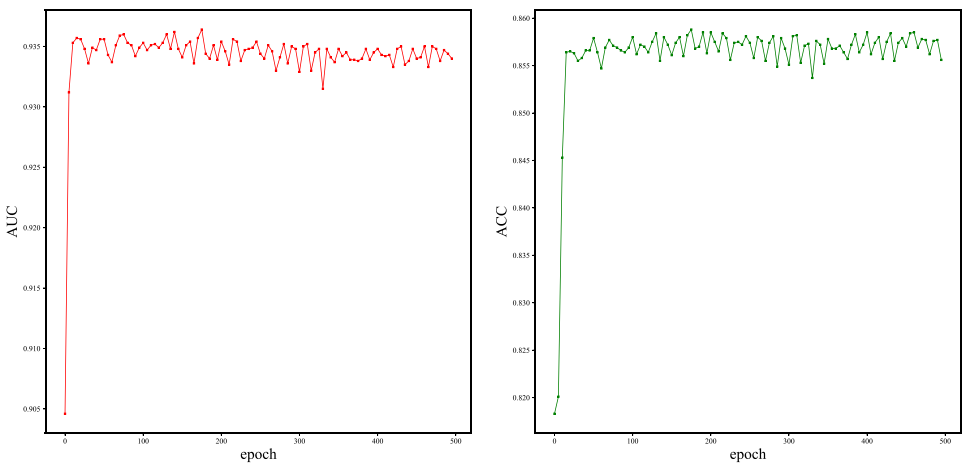
5.6.2 Impact of number of heads and layers

We consider the number of *heads* from {1, 2, 3, 4, 5} and the value of h from {1, 2}. Figure 8 (c) and (d) show that MTRDKG performs best when the value of *heads* is 2, after which the performance gradually declines. When h is 1, MTRDKG shows better performance under the *AUC* metric. However, the model performs better under the *AP* metric when h is 2. We set h to 1 because, with an increase in h, the performance improvement is not significant, but the number of neighboring nodes significantly increases, severely affecting computational efficiency.

5.6.3 Impact of ISU layers and training frequency

We consider the value of L from $\{1, 2, 3, 4, 5\}$ and the value of tim from $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Figure 8 (e) and (f) show

Fig. 7 Performance trend of MTRDKG at each epoch



(a) The change in AUC for each epoch (b) The change in ACC for each epoch

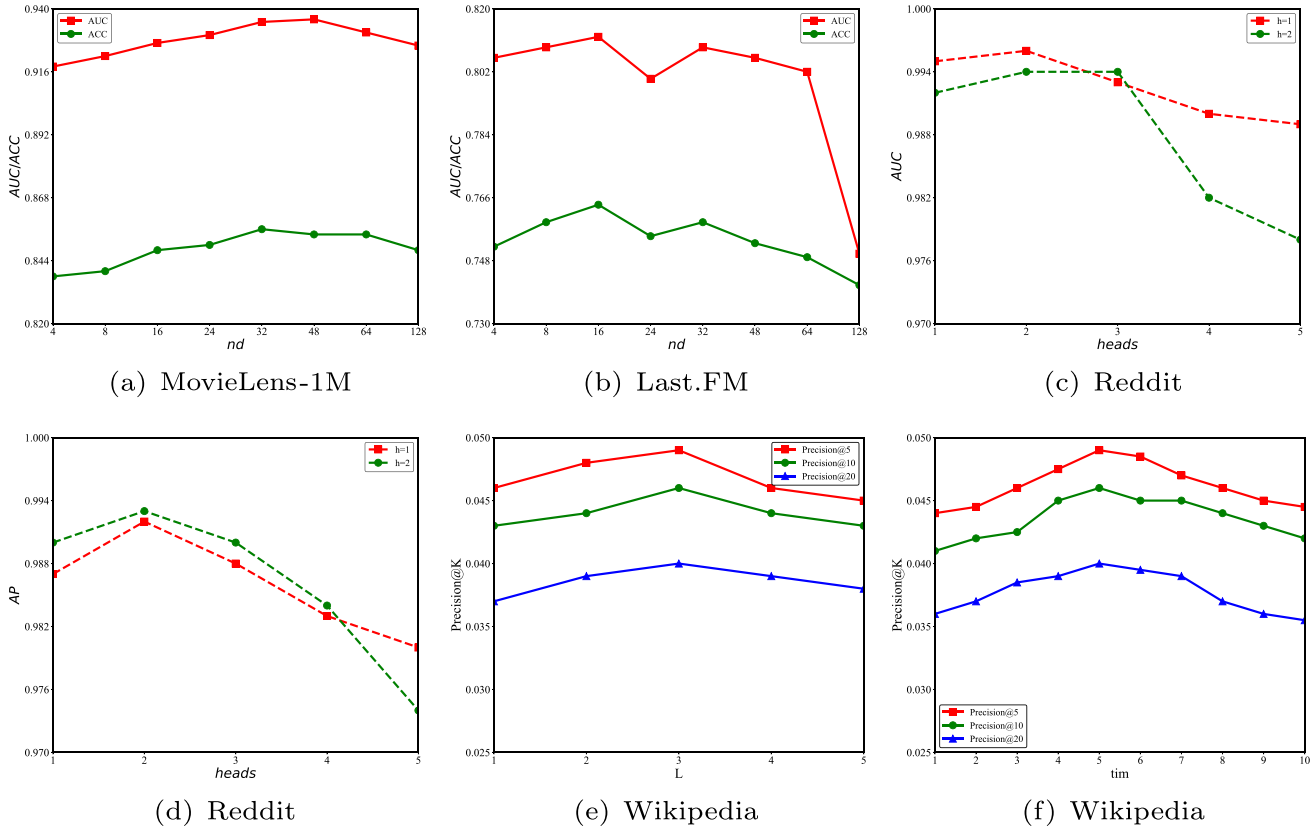


Fig. 8 Parameter Sensitivity Analysis Results

that MTRDKG achieves the best performance when L is 2, as too few layers lead to insufficient information fusion. At the same time, too many layers might integrate some irrelevant information, causing negative transfer. The optimal training frequency is 5, possibly because a training frequency that is too high can cause the objective function value to fluctuate or oscillate, ultimately leading to model overfitting, while a low frequency results in slow model convergence.

6 Summary and future work

The proposed MTRDKG model leverages TGN to model DKG as a series of continuous-time events, allowing the model to fully utilize the rich semantic information in KG while better understanding and mining the patterns of nodes and edges over time. Extensive experiments on four real datasets show that the MTRDKG model significantly outperforms the state-of-the-art baseline methods. Additionally, we observed that the MTRDKG model exhibits excellent performance in both sparse and large-scale data scenarios, indicating the model's robustness, generalization ability, scalability, and efficiency, effectively addressing RS's sparsity and cold start problems. In future work, we will further optimize and improve the combination of recommended items and node embeddings to enhance the model's performance and generalization ability. Moreover, we plan to explore the potential of combining more sophisticated dynamic graph models with multitask feature learning methods to expand the research scope.

Acknowledgements This work was supported by the National Natural Science Foundation of China (No. 62273170), the Surface Project of Liaoning Provincial Department of Education (No. JYTMS20230869), the Scientific Research Project of Liaoning Provincial Department of Education (Nos. JZL202015404, LJKZ0625), and the Liaoning Provincial Higher Education Innovation Talent Support Project (No. LR2019034).

Author Contributions Minwei Wen: Conceptualization, Methodology, Resources, Validation, Data Curation, Writing- Original Draft, Writing - Review and Editing. Hongyan Mei: Methodology, Software, Formal analysis, Data Curation, Writing - Review and Editing. Wei Wang, Xiaorong Xue and Xing Zhang: Formal analysis, Validation, Writing-Review and Editing.

Data Availability All links to data generated or analysed in this study are included in this published article.

Declarations

Conflict of interest We declare that the authors do not have any conflict of interest related to the content of this article.

Ethics approval This article contains no studies with human participants or animals performed by authors.

References

1. Wang Q, Mao Z, Wang B, Guo L (2017) Knowledge graph embedding: A survey of approaches and applications. IEEE Trans Knowl Data Eng 29(12):2724–2743
2. Sang L, Xu M, Qian S, Wu X (2021) Knowledge graph enhanced neural collaborative recommendation. Expert Syst Appl 164:113992
3. Wang H, Zhao M, Xie X, Li W, Guo M (2019) Knowledge graph convolutional networks for recommender systems. In: The world wide web conference, pp 3307–3313
4. Guo Q, Zhuang F, Qin C, Zhu H, Xie X, Xiong H, He Q (2020) A survey on knowledge graph-based recommender systems. IEEE Trans Knowl Data Eng 34(8):3549–3568
5. Zhang F, Yuan NJ, Lian D, Xie X, Ma W-Y (2016) Collaborative knowledge base embedding for recommender systems. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 353–362
6. Lin Y, Liu Z, Sun M, Liu Y, Zhu, X (2015) Learning entity and relation embeddings for knowledge graph completion. In: Proceedings of the AAAI conference on artificial intelligence, vol 29
7. Zhou Z, Wang C, Feng Y, Chen D (2022) Jointe: Jointly utilizing 1d and 2d convolution for knowledge graph embedding. Knowl-Based Syst 240:108100
8. Baghershahi P, Hosseini R, Moradi H (2023) Self-attention presents low-dimensional knowledge graph embeddings for link prediction. Knowl-Based Syst 260:110124
9. Zhang T, Tian X, Sun X, Yu M, Sun Y, Yu G (2021) Overview on knowledge graph embedding technology research. J Softw 34(1):277–311
10. Lin Q, Mao R, Liu J, Xu F, Cambria E (2023) Fusing topology contexts and logical rules in language models for knowledge graph completion. Inf Fus 90:253–264
11. Zhang Y, Yang Q (2021) A survey on multi-task learning. IEEE Trans Knowl Data Eng 34(12):5586–5609
12. Wang H, Zhang F, Zhao M, Li W, Xie X, Guo M (2019) Multi-task feature learning for knowledge graph enhanced recommendation. In: The world wide web conference, pp 2000–2010
13. Ye Q, Hsieh C-Y, Yang Z, Kang Y, Chen J, Cao D, He S, Hou T (2021) A unified drug-target interaction prediction framework based on knowledge graph and recommendation system. Nat Commun 12(1):6775
14. Du Y, Zhu X, Chen L, Fang Z, Gao Y (2022) Metakg: Meta-learning on knowledge graph for cold-start recommendation. IEEE Trans Knowl Data Eng
15. Huang Z, Liu Y, Zhan C, Lin C, Cai W, Chen Y (2021) A novel group recommendation model with two-stage deep learning. IEEE Trans Syst Man Cybern: Syst 52(9):5853–5864
16. Rossi E, Chamberlain B, Frasca F, Eynard D, Monti F, Bronstein M (2020) Temporal graph networks for deep learning on dynamic graphs. Preprint arXiv:2006.10637
17. Qin M, Zhang C, Bai B, Zhang G, Yeung D-Y (2023) High-quality temporal link prediction for weighted dynamic graphs via inductive embedding aggregation. IEEE Trans Knowl Data Eng
18. Wu T, Khan A, Yong M, Qi G, Wang M (2022) Efficiently embedding dynamic knowledge graphs. Knowl-Based Syst 250:109124
19. Zhou L, Yang Y, Ren X, Wu F, Zhuang Y (2018) Dynamic network embedding by modeling triadic closure process. In: Proceedings of the AAAI conference on artificial intelligence, vol 32
20. Goyal P, Chhetri SR, Canedo A (2020) dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowl-Based Syst 187:104816
21. Paudel R, Huang HH (2022) Pikachu: Temporal walk based dynamic graph embedding for network anomaly detection. In: NOMS 2022-2022 IEEE/IFIP network operations and management symposium, pp 1–7. IEEE
22. 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
23. Nguyen GH, Lee JB, Rossi RA, Ahmed NK, Koh E, Kim S (2018) Continuous-time dynamic network embeddings. In: Companion proceedings of the the web conference 2018, pp 969–976
24. Khoshraftar S, Mahdavi S, An A, Hu Y, Liu J (2019) Dynamic graph embedding via lstm history tracking. In: 2019 IEEE international conference on data science and advanced analytics (DSAA), pp 119–127. IEEE
25. Zuo Y, Liu G, Lin H, Guo J, Hu X, Wu J (2018) Embedding temporal network via neighborhood formation. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp 2857–2866
26. Pareja A, Domeniconi G, Chen J, Ma T, Suzumura T, Kanezashi H, Kaler T, Schardl T, Leiserson C (2020) Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In: Proceedings of the AAAI conference on artificial intelligence, vol 34, pp 5363–5370
27. Xu D, Ruan C, Korpeoglu E, Kumar S, Achan K (2020) Inductive representation learning on temporal graphs. In: International conference on learning representations. https://openreview.net/forum?id=rJeW1yHYwH
28. Vandenhende S, Georgoulis S, Van Gansbeke W, Proesmans M, Dai D, Van Gool L (2021) Multi-task learning for dense prediction tasks: A survey. IEEE Trans Pattern Anal Mach Intell 44(7):3614–3633
29. Wang Y, Ding W, Zhang R, Li H (2020) Boundary-aware multitask learning for remote sensing imagery. IEEE J Sel Top Appl Earth Obs Remote Sens 14:951–963
30. Yang E, Pan J, Wang X, Yu H, Shen L, Chen X, Xiao L, Jiang J, Guo G (2023) Adatask: A task-aware adaptive learning rate approach to multi-task learning. In: Proceedings of the AAAI conference on artificial intelligence, vol 37, pp 10745–10753
31. Wang Y, Zhang J, Zhou X, Zhang Y (2022) Hierarchical aggregation based knowledge graph embedding for multi-task recommendation. In: Asia-Pacific Web (APWeb) and web-age information management (WAIM) joint international conference on web and big data, pp 174–181. Springer
32. Hu B, Ye Y, Zhong Y, Pan J, Hu M (2022) Transmkr: Translation-based knowledge graph enhanced multi-task point-of-interest recommendation. Neurocomputing 474:107–114
33. Zhu J, Zhang Y, Wang Y, Liao W, Chen R, Yuan M (2023) Knowledge-enhanced multi-task recommendation in hyperbolic space. Appl Intell 53(23):28694–28710
34. Zhai H, Zheng W, Ouyang Y, Pan X, Zhang W (2024) Multi-focus image fusion via interactive transformer and asymmetric soft sharing. Eng Appl Artif Intell 133:107967
35. Huang W, Wu J, Song W, Wang Z (2022) Cross attention fusion for knowledge graph optimized recommendation. Appl Intell 1–10
36. Gao M, Li J-Y, Chen C-H, Li Y, Zhang J, Zhan Z-H (2023) Enhanced multi-task learning and knowledge graph-based recommender system. IEEE Trans Knowl Data Eng
37. Zhou Y, Guo J, Song B, Chen C, Chang J, Yu FR (2022) Trust-aware multi-task knowledge graph for recommendation. IEEE Trans Knowl Data Eng

38. Shu H, Huang J (2023) Multi-task feature and structure learning for user-preference based knowledge-aware recommendation. Neurocomputing 532:43–55
39. Zeb A, Saif S, Chen J, Haq AU, Gong Z, Zhang D (2022) Complex graph convolutional network for link prediction in knowledge graphs. Expert Syst Appl 200:116796
40. Kazemi SM, Goel R, Eghbali S, Ramanan J, Sahota J, Thakur S, Wu S, Smyth C, Poupart P, Brubaker M (2020) Time2vec: Learning a vector representation of time
41. Pennebaker JW, Francis ME, Booth RJ (2001) Linguistic inquiry and word count: Liwc 2001. Mahway: Lawrence Erlbaum Associates 71(2001), p 2001
42. Cheng H-T, Koc L, Harmsen J, Shaked T, Chandra T, Aradhye H, Anderson G, Corrado G, Chai W, Ispir M et al (2016) Wide & deep learning for recommender systems. In: Proceedings of the 1st workshop on deep learning for recommender systems, pp 7–10
43. Wang H, Zhang F, Wang J, Zhao M, Li W, Xie X, Guo M (2018) Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In: Proceedings of the 27th ACM international conference on information and knowledge management, pp 417–426
44. Zhang Y, Yuan M, Zhao C, Chen M, Liu X (2022) Aggregating knowledge-aware graph neural network and adaptive relational attention for recommendation. Appl Intell 52(15):17941–17953
45. Li H, Li C, Feng K, Yuan Y, Wang G, Zha H Robust knowledge adaptation for dynamic graph neural networks. Preprint arXiv:2207.10839
46. Zhang M, Wu S, Yu X, Liu Q, Wang L (2022) Dynamic graph neural networks for sequential recommendation. IEEE Trans Knowl Data Eng 35(5):4741–4753

Minwei Wen is currently pursuing a master's degree in computer technology at Liaoning University of Technology. His main research interests include multi-task recommendation systems, knowledge graphs, and data mining.

Hongyan Mei Ph.D., is currently a full-time professor at Liaoning University of Technology. Her main research interests include peer-to-peer networks and services, data mining, and big data analytics.

Wei Wang received the B.S. degree and M.S. degrees in control theory and control engineering from Liaoning University of Technology, Jinzhou, China, in 2003 and 2006, respectively, and the Ph.D. degree in control theory and control engineering from Dalian Maritime University, Dalian, China, in 2015. She is currently an associate Professor with the School of Electrical Engineering, Liaoning University of Technology. Her research focuses on intelligent control and cooperative control of multi-agent systems.

Xiaorong Xue Ph.D., is currently a full-time professor at Liaoning University of Technology. His main research interests include image processing and pattern recognition.

Xing Zhang Ph.D., is currently a full-time professor at Liaoning University of Technology. His main research interests include privacy protection, image steganography, blockchain, and information security.