TESA: A Trajectory and Semantic-aware Dynamic Heterogeneous Graph Neural Network

Anonymous Author(s)*

Abstract

Dynamic graph neural networks (DGNNs) are designed to capture the dynamic evolution of graph node interactions. However, existing DGNNs mainly consider homogeneous graphs, neglecting the rich heterogeneity in node and edge types, which is prevalent for real-world graphs and essential for modeling complex dynamic interactions. In this work, we propose the **T**raj**E**ctory and **S**emantic-**A**ware dynamic heterogeneous graph neural network (**TeSa**), which integrates *trajectory-based evolution* and *semantic-aware aggregation* to capture both the evolving dynamics and heterogeneous semantics entailed in continuous-time dynamic heterogeneous graphs. In particular, trajectory-based evolution treats the interactions received by each node (called node trajectory) as a sequence and employs a temporal point process to learn the dynamic evolution in these interactions. Semantic-aware aggregation separates edges of different types when aggregating messages for each node from its neighbors. Edges of the same type are processed at first (i.e., intra-semantic aggregation), and then edges of different types are handled (i.e., inter-semantic fusion), to offer a comprehensive view of the heterogeneous semantics. We compare **TeSa** with 7 state-of-the-art DGNN models, and the results show that **TeSa** improves the best-performing baseline by an average of 5.11% and 5.74% in accuracy for transductive and inductive tasks.

Keywords

dynamic graph, heterogeneous graph, graph learning

ACM Reference Format:

Anonymous Author(s). 2018. TESA: A Trajectory and Semantic-aware Dynamic Heterogeneous Graph Neural Network. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Dynamic heterogeneous graphs (DHGs) serve as a natural abstraction for modeling real-world complex systems, capturing the intricate interactions among diverse entities. The evolution of these networks often reveals underlying patterns. For instance, in Figure 1, user u_1 forwards blog b_2 posted by user u_2 at time t_8, and u_1 has previously followed user u_2, reflecting a triadic closure[22]. Researchers have leveraged such patterns to model system dynamics and enhance predictive capabilities across various domains, such as recommendation systems in social networks [32], over-prescribing prediction in healthcare [27], and route optimization in intelligent transportation systems [11, 20].

Table 1: Anatomy of GNN models. We use *dynamics*, *heterogeneity*, *long-term*, and *memory efficiency* to indicate whether a model learns temporal dynamics, handles heterogeneous types, captures long-term semantic evolution, and avoids excessive memory consumption, respectively.

Model	*Dynamics*	*Heterogeneity*	*Long-term*	*Memory Efficiency*
GAT [23]	✗	✗	✗	✓
HGT [8]	✗	✓	✗	✓
TGAT [1]	✓	✗	✗	✓
TGN [19]	✓	✗	✓	✗
HTGNN [3]	✓	✓	✗	✓
TeSa(ours)	✓	✓	✓	✓

The advent of dynamic graph neural networks (DGNNs) has significantly enhanced our ability to model evolving systems. Nonetheless, a substantial proportion of current research remains concentrated on *dynamic homogeneous* graphs, in which both nodes and edges belong to a single type. This simplification can lead to suboptimal performance as it neglects the underlying semantic information of nodes and edges. Therefore, representation learning on dynamic heterogeneous graphs is crucial for enhancing the modeling capabilities of complex systems. However, it is still in its nascent stages and encounters the following two primary challenges.

Challenge 1: *How can we effectively capture the entangled evolving dynamics and structural heterogeneity entailed in DHGs?* Early approaches [3, 25, 30] conceptualize dynamic heterogeneous graphs as a sequence of graph snapshots, known as discrete-time dynamic heterogeneous graphs (DTDHGs). These methods normally perform heterogeneous message passing on the snapshots and subsequently employ sequence models to capture the temporal evolution within these slices. However, they treat temporal and heterogeneous information in isolation, making them unable to capture the dynamic changes in intricate semantic structures among various entities, and thus failing to represent the intertwined dynamics of these dimensions [21]. Recent works [34] have attempted to integrate temporal and heterogeneous information simultaneously. Nevertheless, the discrete-time framework is still susceptible to information loss and struggles to adequately model the continuous nature of entity interactions. More recently, continuous-time dynamic heterogeneous graphs (CTDHGs) have been explored [10], which offer more precise temporal modeling by representing DHG as a continuous event sequence. However, [10] relies on a self-exciting process to model the evolution of the graph, which assumes that past events always stimulate the occurrence of current events. Thus, the model struggles to capture the complex interdependencies and diverse influences among events within the graph.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

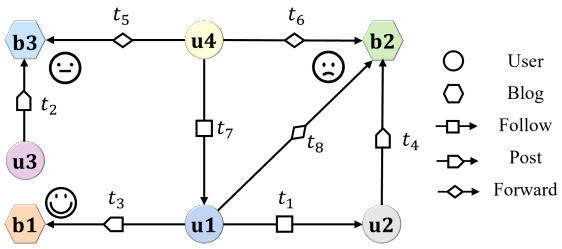


Figure 1: A toy example of dynamic heterogeneous social network, where the node shapes indicate node types, the symbols between arrows represent edge types, and each emoji represents the sentiment expressed in a blog.

Challenge 2: *How can we capture node's robust intrinsic semantics and behaviors over time while supporting large-scale graphs?* As illustrated in Figure 1, three blogs discuss different stances on a particular event: blog b_1 (supportive), blog b_2 (opposed), and blog b_3 (neutral). User u_4 has historically shown a neutral to somewhat opposed position, initially sharing blog b_3 (neutral) and then forwarding blog b_2 (opposed). Although user u_4 follows user u_1 (supportive, posting blog b_1) at time t_7 due to other reasons, this does not necessarily imply that user u_4 will share blog b_1, given its past interactions primarily with neutral and opposed content. As can be observed, such long-term, while robust, behaviors are often overlooked by previous work [1, 26], as they tend to focus more on the aggregation of neighboring information and neglect the change of node semantics over time, leading to a loss of historical context. Although some works [13, 19] have employed a global memory module to store historical representations of nodes. However, these works have not explicitly investigated behavior evolution over time and the global module limits the scalability of these models in the presence of large-scale graphs.

To address the aforementioned challenges, we propose the *TrajEctory and Semantic-Aware Dynamic Heterogeneous Graph Neural Network* (**TeSa**[1]), which models dynamic heterogeneous graphs from a continuous-time perspective. The **TeSa** layer integrates two key components: trajectory-based evolution and semantic-aware aggregation, together effectively capturing the entangled dynamics and heterogeneous semantics in DHGs. The *trajectory-based evolution* module is crucial for addressing the challenge of capturing a node's intrinsic semantics and behaviors over time. Specifically, it extracts a trajectory for each node, which is a sequence of temporally-ordered events associated with the node. However, the trajectory can be rather sophisticated, as each point captures the timing, type, and features of the present interaction. To tackle this problem, we adopt the technique of temporal point processes to elegantly encode these sophisticated trajectories, thus effectively capturing the dynamic semantic changes. Building upon the outputs of the trajectory evolution module, the *semantic aggregation* module consolidates the topological and semantic relationships among nodes. It first performs intra-semantic aggregation to integrate information from neighboring nodes based on various edge types, followed by inter-semantic fusion, which fuses the above aggregated representations to create a cohesive view of the heterogeneous semantics. Ultimately, the final embedding for each layer is generated by combining the trajectory-evolved node embedding with the neighboring aggregated semantic representation. Besides, **TeSa** does not introduce any global memory module, therefore ensuring higher scalability compared to previous works [13, 19].

[1]https://anonymous.4open.science/r/TeSa-45FF

Table 2: Notations

Symbol	Description
$\mathbf{h}_i^l(t)$	Hidden embedding of node i at layer l and time t
$\mathbf{h}_i^{l,tr}(t)$	Trajectory-evolved embedding of node i at layer l and time t
$\mathbf{h}_{i,r}^l(t)$	Intra-semantic aggregated embedding of node i for relation type r at layer l and time t
$\mathbf{h}_{i,R}^l(t)$	Semantic-fused embedding of node i at layer l and time t across all the edge types in set $\mathcal{R}$

We evaluate **TeSa** and stat-of-the-art baselines over various datasets. **TeSa** consistently outperforms the baselines; for example, **TeSa** achieves AUC improvements of 4.2% and 6.2% in transductive and inductive link prediction on the Yelp(C) dataset, respectively. Furthermore, we conduct ablation experiments to validate the effectiveness of each component of our model.

In summary, our technical contributions are threefold:

- We propose **TeSa**, a framework for dynamic heterogeneous graph representation learning, which takes a continuous-time view and accurately captures the intertwined dynamics and heterogeneous semantics inherent in DHGs.
- **TeSa** can capture the dynamic changes of node behaviors by employing temporal point process to encode trajectories. This approach enables effective modeling of evolving semantics of nodes, thereby alleviating challenges associated with dynamic semantic changes in heterogeneous contexts.
- Our empirical evaluations demonstrate that **TeSa** significantly outperforms the existing methods, highlighting its effectiveness and robustness in various scenarios.

2 Preliminary

In this section, we introduce the notations and problem definition. We first define a dynamic heterogeneous graph as a sequence of events from a continuous-time perspective.

Definition 2.1 (Dynamic Heterogeneous Graph, DHG). A dynamic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{A}, \mathcal{R}, \mathcal{X})$ is characterized by a set of nodes $\mathcal{V}$, a set of edges $\mathcal{E}$, a time domain $\mathcal{T}$, a set of node types $\mathcal{A}$, a set of edge types $\mathcal{R}$, and an input feature set $\mathcal{X} = \{\mathbf{X}_v, \mathbf{X}_e\}$, where $\mathbf{X}_v$ and $\mathbf{X}_e$ denote node and edge features, respectively. Besides, the graph is associated with two type mapping functions: $\phi : \mathcal{V} \to \mathcal{A}$ and $\Phi : \mathcal{E} \to \mathcal{R}$. For each node $v \in \mathcal{V}$, $\mathbf{x}_v \in \mathbb{R}^{d_v}$ denotes its feature vector, where d_v is related to the node type determined by $\phi(v)$. Each edge $e = (v_i, v_j, t)$ represents an event occurring at time t, indicating an interaction between nodes v_i and v_j. The edge feature $\mathbf{x}_e \in \mathbb{R}^{d_e}$ is defined analogously, with the dimension d_e determined by the edge type $\Phi(e)$. When $|\mathcal{A}| + |\mathcal{R}| > 2$, $\mathcal{G}$ is classified as a dynamic heterogeneous graph.

Notably, if $|\mathcal{A}| + |\mathcal{R}| = 2$, it is considered as a dynamic homogeneous graph. In this context, we define the problem as follows.

Definition 2.2 (Representation Learning on DHG). Given a dynamic heterogeneous graph $\mathcal{G}$, representation learning on DHGs aims to learn an embedding function $\mathbf{h} : \mathcal{V} \times \mathcal{T} \to \mathbb{R}^D$ that maps each node $v \in \mathcal{V}$ at any timestamp $t \in \mathcal{T}$ to a D-dimensional embedding vector $\mathbf{h}_v(t)$.

The learned time-aware node embeddings should effectively encapsulate the evolving dynamics and heterogeneous semantics inherent in $\mathcal{G}$, thereby facilitating various downstream tasks such as link prediction [12, 16], node classification [10, 16], and node regression [3, 34]. Next, we define the node trajectory in DHGs that we want to learn in this work.

Definition 2.3 (Node Trajectory). Given a dynamic heterogeneous graph $\mathcal{G}$, the trajectory of a node v is a temporally-ordered sequence of events in which the node v participates, which is represented as:

$$\mathcal{T}_v = \{(e_1, e_2, \ldots, e_n) \mid e_i \in \mathcal{E}_v\}$$

where $\mathcal{E}_v$ denotes the set of events associated with node v, and $n = |\mathcal{E}_v(t)|$ is the total number of events in the trajectory. In dynamic heterogeneous graph, each event e_i is formalized as the tuple $e_i = (t_i, k_i, \mathbf{x}_i)$, where t_i denotes the timestamp, $k_i = \Phi(e_i)$ indicates the event type, and $\mathbf{x}_i$ represents the event feature.

This trajectory naturally reflects the evolving semantics and behaviors of node v over time. And we summarize the frequently used symbols related to our model in Table 2.

3 Methodology

3.1 Overview

Our proposed model **TeSa** serves as a representation learning framework for continuous-time dynamic heterogeneous graphs. Due to the variety of dynamic heterogeneous graphs, it is essential to convert raw node features into a unified feature space. Specifically, for a node v with raw attributes $\mathbf{x}_v$, the transformed attribute is defined as $\mathbf{x}'_v = M_{\phi(v)} \cdot \mathbf{x}_v$, where $M_{\phi(v)}$ is a *learned type-specific* transformation matrix. This process is similarly applied to raw edge features. Each **TeSa** layer processes the node embeddings via two fundamental phases, as depicted in Figure 2.

(1) **Trajectory-based Evolution.** This step constructs historical trajectory for each node, where each point in the trajectory corresponds to an event involving the node. By utilizing the neural temporal point process (NTPP), this component captures the complex dependency and temporal evolution of events in the node's sequence. This approach effectively models how node behavior and semantics change over time, leading to the trajectory-evolved embedding $\mathbf{h}_v^{l,tr}(t)$. In Section 3.2, we will elaborate on how to model trajectory-based evolution with NTPP.

(2) **Semantic-aware Aggregation.** This stage involves two key processes: intra-semantic aggregation and inter-semantic fusion. Intra-semantic aggregation focuses on gathering information from neighboring nodes according to different edge types, allowing the model to perceive specific semantic relationships. The output of this process is then subjected to inter-semantic fusion, which fuses the aggregated information to produce a comprehensive representation of the node's contextual relationships, denoted as $\mathbf{h}_{v,R}^l(t)$. In Section 3.3, we will introduce the details of semantic-aggregation.

Finally, the trajectory-evolved embedding $\mathbf{h}_v^{l,tr}(t)$ is combined with the semantic-fused neighbor information $\mathbf{h}_{v,R}^l(t)$. This integration enhances the node's representation by incorporating both its historical dynamics and the semantic relationships derived from its neighbors, resulting in the final node embeddings $\mathbf{h}_v^l(t)$ for the current layer. By stacking multiple **TeSa** layers, our model can effectively capture the intertwined evolving dynamics and heterogeneous semantics within DHGs, yielding comprehensive representations that facilitate accurate analysis and prediction.

3.2 Trajectory-based Evolution

The trajectory-based evolution module captures the trajectory of each node, i.e., the historical interaction sequences, which is crucial for accurately modeling the evolving behaviors and inherent semantics of nodes over time.

Trajectory Construction. The trajectory of a node v up to time t is constructed by recording the sequence of events, which is represented as:

$$\mathcal{T}_v(t) = \{e_i \mid e_i \in \mathcal{T}_v, t_i < t\} \tag{1}$$

where $\mathcal{T}_v$ is the trajectory of node v, as defined in Definition 2.3. Each event e_i is modeled as a tuple $e_i = (t_i, k_i, \mathbf{c}_i)$, where t_i and k_i represent the timestamp and event type as previously defined. The difference lies in $\mathbf{c}_i$, which represents the contextual feature vector of the event. We consider $\mathbf{c}_i$ as the average embedding of the nodes associated with event e_i from the previous layer, which allows the model to dynamically incorporate contextual information from neighboring interactions. This approach reflects real-time changes in node behavior, as it incorporates semantic information from neighboring nodes, thereby better capturing the dynamics of interactions compared to static event features.

Trajectory Sampling. To facilitate efficient batch processing, we propose a *type-aware temporal-biased sampling* strategy to capture both semantic and temporal relevance. Specifically, given a node's trajectory $\mathcal{T}_v(t)$, we aim to sample a sub-trajectory $\hat{\mathcal{T}}_v(t)$ with a fixed length l.

Step 1: Type-aware Division. We divide $\mathcal{T}_v(t)$ into subsets based on event types:

$$\mathcal{T}_v(t) = \bigcup_k \mathcal{T}_v^k(t) \tag{2}$$

where $\mathcal{T}_v^k(t)$ represents the subset of events with type k. For each subset $\mathcal{T}_v^k(t)$, we sample events proportionally to its length relative to the total trajectory length $|\mathcal{T}_v(t)|$, ensuring that the sampled trajectory retains event-type diversity. The number of events sampled from subset $\mathcal{T}_v^k(t)$ is determined by:

$$l_k = \left\lfloor \frac{|\mathcal{T}_v^k(t)|}{|\mathcal{T}_v(t)|} \times l \right\rfloor \tag{3}$$

where $|\mathcal{T}_v^k(t)|$ is the number of events of type k, and l_k represents the number of events sampled from $\mathcal{T}_v^k(t)$.

Step 2: Time-sensitive Sampling. Within each subset $\mathcal{T}_v^k(t)$, the sampling strategy considers the event timestamps, ensuring that

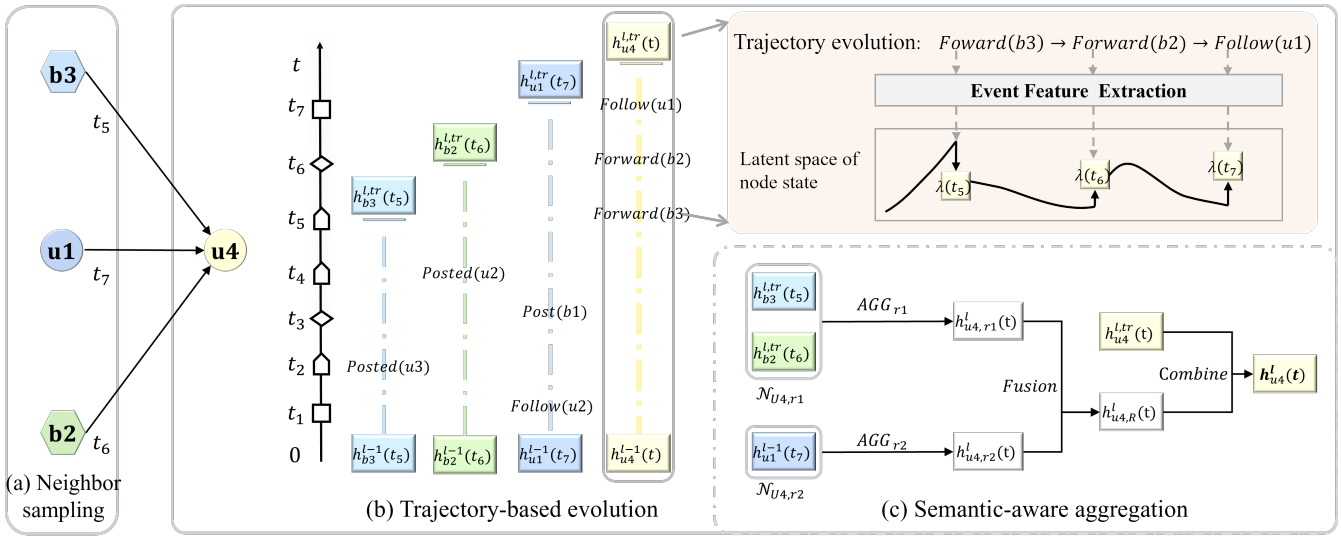


Figure 2: Workflow of the l^{th} TeSa layer for user u_4 at time t. (a) For u_4, its neighbors prior to time t are sampled, resulting in u_1, b_2, and b_3. (b) Trajectory-based evolution, which constructs trajectories for user u_4 and its sampled neighbors. For instance, in the rightmost dashed yellow line, Forward(b3) indicates that u_4 forwards blog b_3 at time t_5. The trajectory is then processed to obtain the trajectory-evolved embedding $\mathbf{h}^{(l,tr)}_{u_4}(t)$, as shown in the orange block in the upper right corner. Each event passes through an event feature extraction module, then the output is mapped to the node's latent embedding space. (c) Semantic-aware aggregation. Intra-semantic aggregation aggregates trajectory-evloved embeddings within each edge type (e.g., $r1$ and $r2$ representing forward and follow, respectively), followed by inter-semantic fusion that produces a holistic representation of across different edge types. Finally, the above output is combined with the trajectory-evolved embedding of the target node to generate the output embedding $\mathbf{h}^{l}_{u_4}(t)$ of the l^{th} layer.

more recent events are prioritized[26]. The probability of sampling an event e_i is given by:

$$P(e_i) = \frac{\exp(t - t_i)}{\sum_{e_j \in \mathcal{T}^k_v(t)} \exp(t - t_j)} \tag{4}$$

However, this sampling strategy introduces additional efficiency burdens due to its computational complexity. To mitigate this, we simplify the process by selecting only the most recent event within each type-specific subset, thereby retaining key temporal information while significantly reducing computational complexity.

Trajectory Evolution. To effectively model the evolution of node trajectories, it is essential for the encoder to satisfy two primary requirements: 1) capture the complex dependencies between events within a trajectory, thereby reflecting the changes in the node's state; and 2) jointly model both event types and timestamps, facilitating the encoding of dynamic node semantics.

Fortunately, the Neural Temporal Point Process (NTPP) serves as an excellent framework for handling such marked event sequences. Its core concept lies in leveraging neural networks to capture the temporal and categorical dependencies within the event sequence. Given an event sequence of length l, denoted as $\{e_1, e_2, \ldots, e_l\}$, the neural network takes this sequence as input and generates a representation $\mathbf{h}_l$, which encapsulates the evolving information from the events:

$$\mathbf{h}_l = f_{\text{NN}}(e_1, e_2, \ldots, e_l) \tag{5}$$

where f_{NN} is a neural network model that captures the dependencies and patterns inherent in the sequence. This output embedding $\mathbf{h}_l$ can then be used to predict future events.

In our research, the sequence of events is constructed from the historical events of a node, thereby effectively reflecting the changes in the node's semantics. Motivated by this, *we define $\lambda_v(t)$ as the evolving trajectory embedding for node v*, which captures the dynamic changes in its semantics and behavior over time. To achieve this, we propose a NTPP-based trajectory evolution module, *tra_evo*, which can be divided into the following two steps.

Step 1: Event Feature Extraction. Given a sampled trajectory $\hat{\mathcal{T}}_v(t)$ consisting of l events, we define the event feature vector $\mathbf{z}_i = [\mathbf{c}_i, \tilde{t}_i, \tilde{k}_i]$, where $\mathbf{c}_i$ represents the contextual feature, $\tilde{t}_i$ is the encoded time embedding, and $\tilde{k}_i$ is a learned embedding of the event type. The time embedding $\tilde{t}_i$ is encoded as follows:

$$\tilde{t}_{i,d} = \begin{cases} \sin\left(\frac{t_i}{10000^{d/D}}\right), & \text{if } d \text{ is even} \\ \cos\left(\frac{t_i}{10000^{(d-1)/D}}\right), & \text{if } d \text{ is odd} \end{cases} \tag{6}$$

where $0 \le d < D$ is the time embedding dimension. This design allows each learnable module to directly access the individual components of the event, thereby enhancing the model's flexibility in capturing diverse event dynamics.

Step 2: Evolution Process. The implementation of NTPP offers a variety of architectures to model the evolution of event sequences[31].

In this work, we adopt an attention-based architecture, and the exploration of additional architectures can be found in Section 4.3.

In this method, each event e_i in the sequence is treated as a query, key, and value. The query $\mathbf{q}_i$, key $\mathbf{k}_i$, and value $\mathbf{v}_i$ for each event e_i are calculated using learnable weight matrices:

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{z}_i, \quad \mathbf{k}_i = \mathbf{W}_k \mathbf{z}_i, \quad \mathbf{v}_i = \mathbf{W}_v \mathbf{z}_i \tag{7}$$

where $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ are learnable weight matrices that project the event feature vector $\mathbf{z}_i$ into the query, key, and value spaces, respectively. The trajectory embedding $\lambda_v(t_i)$ for event e_i can be obtained as follows:

$$\lambda_v(t_i) = \mathbf{z}_i + \tanh\left(\sum_{e_j \in \mathcal{H}(e_i)} \frac{\mathbf{v}_j \alpha(e_j, e_i)}{1 + \sum_{e_j \in \mathcal{H}(e_i)} \alpha(e_j, e_i)} \right) \tag{8}$$

where $\mathcal{H}(e_i)$ denotes the set of historical events preceding e_i. The unnormalized attention weight $\alpha(e_j, e_i)$ is calculated as:

$$\alpha(e_j, e_i) = \exp\left(\frac{\mathbf{k}_j^\top \mathbf{q}_i}{\sqrt{D}} \right) \tag{9}$$

Finally, the trajectory-evolved embedding $\mathbf{h}_v^{l,\text{tr}}(t)$ for node v at layer l and time t is given by the embedding at the last time point:

$$\mathbf{h}_v^{l,\text{tr}}(t) = \lambda_v(t_l) \tag{10}$$

where l is the length of our sampled trajectory. In this manner, the trajectory-evolved embedding can effectively integrate information from preceding events, thereby capturing the dynamic changes and semantic features of the trajectory.

Comparison with Existing Methods. Unlike temporal random walk [5, 17] and metapath-based [4, 24] approaches, which rely on complex traversal or costly metapath discovery, our method directly models the node's historical event sequence. This allows us to capture semantic changes without complex neighbor sampling. Furthermore, unlike memory-based methods [7, 19], which rely on global memory modules that limit scalability, our approach dynamically constructs the trajectory of each node, achieving both efficiency and scalability.

3.3 Semantic-aware Aggregation

The semantic-aware aggregation module is intentionally designed to effectively capture the intricate semantic structure inherent in dynamic heterogeneous graphs. By integrating information from neighboring nodes while considering the diverse types of relationships that connect them, this module substantially enhances the representational capabilities of individual nodes. The aggregation process is organized into two principal stages: Intra-Semantic Aggregation and Inter-Semantic Fusion.

Intra-Semantic Aggregation. This stage concentrates on aggregating information from neighbors of *the same type*, allowing the model to capture specific semantic relationships inherent to each type of interaction. Specifically, for each node v at layer l and time t, the neighboring information is aggregated from its neighbors connected by a particular relation type r. The process is defined as follows:

$$\mathbf{h}_{v,r}^l(t) = \text{AGG}_r\left(\left\{ \mathbf{h}_j^{l-1}(t) \mid j \in \mathcal{N}_v^r(t) \right\} \right) \tag{11}$$

where $\mathbf{h}_{v,r}^l(t)$ represents the intra-semantic aggregated embedding of node v for relation type r at layer l and time t, $\mathcal{N}_v^r(t)$ denotes the set of neighbors of node v that are connected through relation type r before time t, and AGG_r is the aggregation function specific to relation type r. This step enables the model to focus on the specific semantics related to each type of relationship, allowing for a more precise representation of neighbor information under a singular semantic context.

The aggregation function AGG_r can be implemented using simple methods such as summation or mean pooling, which aggregate the embeddings of neighboring nodes without accounting for their varying significance. While these approaches are computationally efficient, they often overlook the differing influences of neighbors in dynamic contexts. To address this limitation, we employ a more sophisticated strategy involving an attention mechanism. This allows the model to selectively focus on the most relevant neighbors, not necessarily those most recent in time, but those whose interactions are most significant given the node's current context. Such an attention mechanism ensures that the aggregation process captures both node states and dynamic patterns, leading to a richer and more nuanced node representation.

Inter-semantic Fusion. The second stage integrates the results from the intra-semantic aggregation stage and forms a comprehensive neighboring information representation. Specifically, for each node v, the final embedding is computed as:

$$\mathbf{h}_{v,R}^l(t) = \textit{fusion}\left(\left\{ \mathbf{h}_{v,r}^l(t) \mid r \in \mathcal{R} \right\} \right) \tag{12}$$

where $\mathbf{h}_{v,R}^l(t)$ represents the semantic-fused embedding of node v at layer l and time t, with R indicating that the fusion encompasses embeddings across all edge types in $\mathcal{R}$. The *fusion* function integrates all intra-semantic embeddings from the previous stage, combining them to produce a comprehensive node representation.

We also employ an attention mechanism for the fusion process. This approach computes attention weights for each relation type, enabling the model to assess the relevance of each embedding contextually. By prioritizing significant embeddings, this method enhances the final node representation, ensuring that the integration of diverse semantic aspects from the intra-semantic aggregation stage reflects their importance in the current context.

Final Representation Generation. Finally, we combine the trajectory-aware embedding with the semantic-fused embedding, allowing us to leverage both temporal dynamics and semantic information. The combined node embedding is defined as:

$$\mathbf{h}_v^l(t) = \textit{combine}\left(\mathbf{h}_v^{l,\text{tr}}(t), \mathbf{h}_v^{l,R}(t) \right) \tag{13}$$

where $\mathbf{h}_v^l(t)$ represents the combined embedding of node v at layer l and time t. The function *combine* integrates these two types of embeddings to produce the final node representation, which can be implemented through concatenation followed by a linear transformation or other methods tailored to the needs of the task.

Comparison with Existing Methods. In contrast to existing methods for discrete-time dynamic heterogeneous graphs, which perform semantic aggregation across various snapshots without

accounting for the temporal significance of individual events, our approach effectively incorporates time differences during the aggregation process. This temporal attention mechanism enables the model to selectively emphasize relevant events based on their historical context, capturing fine-grained temporal dynamics that are often overlooked in previous methods [3, 30, 34].

3.4 Model Training

Event Conditional Intensity. After obtaining the embeddings of two nodes u and v at layer L and time t, denoted as $\mathbf{h}_u^L(t)$ and $\mathbf{h}_v^L(t)$, we model the conditional intensity $\lambda_{u,v}(t)$ for an event between these nodes as:

$$\lambda_{u,v}(t) = \sigma\left(\mathrm{FCL}_r((\mathbf{h}_u^L(t) - \mathbf{h}_v^L(t))^2)\right) \tag{14}$$

where FCL_r represents the fully connected layer specific to the edge type r. The input to FCL_r is the element-wise square of the difference between $\mathbf{h}_u^L(t)$ and $\mathbf{h}_v^L(t)$. This differential representation serves as a strong predictor for the occurrence of events between the two nodes. A sigmoid activation function is applied to ensure a positive intensity value.

Loss Function. For any event (u, v, t) that occurs in the graph, we expect a higher conditional intensity $\lambda_{u,v}(t)$, while for non-occurring events, we expect lower values of $\lambda_{u,v}(t)$. Thus, the loss function based on negative log-likelihood is formulated as:

$$L_e(u, v, t) = -\log(\lambda_{u,v}(t)) - \log(1 - \lambda_{u,k}(t)), \tag{15}$$

where k is a negative sample drawn such that the edge (u, k, t) shares the same type as the positive edge (u, v, t). By maintaining the same edge type for both positive and negative samples, the model is compelled to rely more on the node embeddings themselves, thereby enhancing its capability to learn meaningful representations.

Optimization. The overall loss function is defined by considering the set of training events $\mathcal{I}^{tr} = \{(u, v, t) \in \mathcal{I} : t \leq t^{tr}\}$, encompassing all events up to time t^{tr}, as follows:

$$\arg\min_{\Theta} \sum_{(u,v,t)\in\mathcal{I}^{tr}} L_e(u, v, t) + \eta\|\tilde{k}\|^2 \tag{16}$$

where $\eta > 0$ is a hyperparameter controlling the regularization on the learned embedding $\tilde{k}$ of the event type. The regularization term assists in preventing overfitting by constraining the event-type embeddings. For a complete overview of the model training process, please refer to Section D.

4 Experimental Evaluation

4.1 Experiment Settings

Datasets and Baselines. The datasets used in our experiments are summarized in Table 3. We compare our model against seven baseline methods. They can be divided into three categories: (1) Static graph neural networks: GAT [23], HGT [8]; (2) Dynamic homogeneous graph neural networks: TGN [19], TGAT [1], TREND [28]; (3) Dynamic heterogeneous graph neural networks: DyHATR [30], HTGNN [3]. For more details about the datasets and baselines, please refer to the Appendix A and B.

Tasks. We adopt temporal link prediction as our main task, aiming to predict future links based on historical data. Given a dynamic heterogeneous graph, we chronologically split the events into training, validation, and testing sets with a ratio of 75%-15%-15%. Specifically, the training set is defined as $\mathcal{I}^{tr} = \{(u, v, t) \in \mathcal{I} : t \leq t^{\mathrm{tr}}\}$, which consists of all events up to time t^{tr}; the validation set is denoted as $\mathcal{I}^{val} = \{(u, v, t) \in \mathcal{I} : t^{\mathrm{tr}} < t \leq t^{\mathrm{val}}\}$, including events that occur between t^{tr} and t^{val}; and the testing set is defined as $\mathcal{I}^{test} = \{(u, v, t) \in \mathcal{I} : t > t^{\mathrm{val}}\}$, comprising the remaining events occurring after t^{val}. In the inductive setting, we mask 10% of the nodes as unseen during training, removing their associated edges from the training set. Following [9], we define two inductive scenarios: the *new-old* setting predicts edges between a seen node and an unseen node, while the *new-new* setting focuses on edges between two unseen nodes. This allows us to comprehensively assess the model's performance across different link prediction scenarios. We evaluate the model's performance using the Area Under the Curve (AUC) and Average Precision (AP) metrics.

Table 3: Statistics of the experiment datasets, D indicates that a dataset is discrete time while C means continuous time.

Datasets	Nodes	Relations
Aminer (D)	#Author (A): 23,035 #Paper (P): 18,460 #Venue (V): 22	#Publish (P-V): 18,460 #Write (A-P): 52,535 #Cooperate (A-A): 71,680
Yelp (D)	#User (U): 55,702 #Item (I): 12,524	#Review (U-I): 87,846 #Tip (U-I): 35,508
Yelp (C)	#User (U): 1,987,897 #Business (B): 150,346	#Review (U-B): 6,990,247 #Tip (U-B): 908,915

Hyper-parameters. The dimension of hidden embeddings is set to 172 for the Aminer(D) dataset and 32 for both the Yelp(D) and Yelp(C) dataset. The regularization weight η in the loss function is selected as 0.0001. For optimization, we employ the Adam optimizer with a learning rate of 0.0001.

4.2 Main Results

In Table 4, we compare the performance of **TeSa** with the baselines on the temporal link prediction task. In general, our approach demonstrates superior performance, particularly in the inductive new-new setting. Specifically, our model achieves AUC values of **0.8155** on Aminer(D), **0.7612** on Yelp(D), and **0.7908** on Yelp(C) in this setting, surpassing the best baseline model by **9.44%**, **5.67%**, and **8.96%**, respectively. These enhancements can be attributed to our trajectory evolution module, which enables newly introduced nodes to more effectively capture long-term semantic information from their surroundings.

From the perspective of **dynamics**, we observe several key findings. First, dynamic models consistently outperform static models (GAT and HGT). This suggests that leveraging temporal information allows dynamic models to capture the evolving behaviors of nodes, while static models cannot. Second, continuous-time dynamic models surpass discrete-time approaches. We apply a 15-day slicing granularity to both DyHART and HTGNN on the Yelp(C) dataset, and their performance remains poor across various settings. Even on the discrete datasets Aminer(D) and Yelp(D) , their performance is suboptimal in the transductive setting, indicating that discrete

Table 4: Link prediction accuracy for transductive and inductive settings. We use bold font and underline to mark the best and runner-up methods, respectively. Symbol '—' means that a model cannot run due to the out-of-memory (OOM) error. Imp. is the accuracy improvement of TeSa over the best baseline.

Setting		Model	Aminer(D)		Yelp(D)		Yelp(C)	
			AUC	AP	AUC	AP	AUC	AP
Transductive		GAT	0.5541 ± 0.0001	0.5254 ± 0.0001	0.6880 ± 0.0001	0.6941 ± 0.0001	0.6470 ± 0.0017	0.6254 ± 0.0029
		HGT	0.6600 ± 0.0057	0.6917 ± 0.0021	0.6847 ± 0.0004	0.6031 ± 0.0007	0.7069 ± 0.0011	0.6985 ± 0.0007
		TGAT	0.8453 ± 0.0011	0.8216 ± 0.0027	0.7987 ± 0.0022	0.7955 ± 0.0018	0.7721 ± 0.0036	0.7761 ± 0.0058
		TREND	0.8420 ± 0.0035	0.8174 ± 0.0037	0.7978 ± 0.0023	0.7942 ± 0.0022	0.7617 ± 0.0103	0.7654 ± 0.0142
		TGN	0.8197 ± 0.0047	0.7922 ± 0.0052	0.7908 ± 0.0046	0.7709 ± 0.0052	—	—
		DyHART	0.7103 ± 0.0031	0.6532 ± 0.0023	0.7603 ± 0.0023	0.7528 ± 0.0026	0.5744 ± 0.0308	0.5782 ± 0.0420
		HTGNN	0.7259 ± 0.0043	0.6635 ± 0.0024	0.7749 ± 0.0020	0.7649 ± 0.0017	0.5901 ± 0.0798	0.6002 ± 0.0825
		TeSa	**0.9278** ± 0.0097	**0.9144** ± 0.0113	**0.8276** ± 0.0023	**0.8203** ± 0.0018	**0.8141** ± 0.0052	**0.8016** ± 0.0064
		Imp.	8.25%	9.28%	2.89%	2.48%	4.20%	2.55%
Inductive	New-Old	GAT	0.5641 ± 0.0001	0.5488 ± 0.0001	0.6239 ± 0.0001	0.6128 ± 0.0001	0.6853 ± 0.0027	0.6544 ± 0.0041
		HGT	0.6007 ± 0.0020	0.5685 ± 0.0018	0.6575 ± 0.0015	0.6415 ± 0.0021	0.6986 ± 0.0032	0.6776 ± 0.0022
		TGAT	0.6533 ± 0.0112	0.6746 ± 0.0105	0.6872 ± 0.0014	0.6989 ± 0.0010	0.7443 ± 0.0083	0.7502 ± 0.0072
		TREDN	0.6287 ± 0.0166	0.6544 ± 0.0191	0.6875 ± 0.0032	0.6990 ± 0.0026	0.7402 ± 0.0103	0.7409 ± 0.0114
		TGN	0.6309 ± 0.0022	0.6732 ± 0.0034	0.6782 ± 0.0023	0.6741 ± 0.0037	—	—
		DyHATR	0.7213 ± 0.0043	0.6728 ± 0.0018	0.6172 ± 0.0037	0.6031 ± 0.0029	0.6233 ± 0.0125	0.6314 ± 0.0326
		HTGNN	0.7479 ± 0.0052	0.6899 ± 0.0047	0.6377 ± 0.0020	0.6257 ± 0.0017	0.6580 ± 0.1470	0.6570 ± 0.1424
		TeSa	**0.7985** ± 0.0234	**0.7861** ± 0.0173	**0.7065** ± 0.0013	**0.7123** ± 0.0019	**0.7781** ± 0.0025	**0.7807** ± 0.0031
		Imp.	5.06%	9.62%	1.90%	1.33%	3.38%	3.05%
	New-New	GAT	0.5234 ± 0.0002	0.5486 ± 0.0001	0.5762 ± 0.0001	0.5892 ± 0.0001	0.4756 ± 0.0001	0.4996 ± 0.0001
		HGT	0.7211 ± 0.0113	0.7661 ± 0.0035	0.5373 ± 0.0115	0.4565 ± 0.0027	0.6859 ± 0.0000	0.7058 ± 0.0001
		TGAT	0.6154 ± 0.0066	0.6235 ± 0.0061	0.7009 ± 0.0056	0.7118 ± 0.0050	0.7012 ± 0.0083	0.7141 ± 0.0104
		TREND	0.5931 ± 0.0112	0.6021 ± 0.0108	0.7045 ± 0.0055	0.7143 ± 0.0038	0.6937 ± 0.0027	0.7133 ± 0.0042
		TGN	0.5702 ± 0.0165	0.5563 ± 0.0209	0.7036 ± 0.0022	0.6978 ± 0.0016	—	—
		DyHATR	0.6831 ± 0.0025	0.6642 ± 0.0037	0.6013 ± 0.0020	0.5542 ± 0.0023	0.4873 ± 0.0153	0.5133 ± 0.0382
		HTGNN	0.7038 ± 0.0027	0.6856 ± 0.0031	0.6119 ± 0.0027	0.5493 ± 0.0013	0.4468 ± 0.0862	0.4649 ± 0.0404
		TeSa	**0.8155** ± 0.0153	**0.8025** ± 0.0114	**0.7612** ± 0.0025	**0.7660** ± 0.0031	**0.7908** ± 0.0031	**0.8015** ± 0.0037
		Imp.	9.44%	3.64%	5.67%	5.17%	8.96%	8.74%

models struggle to capture fine-grained temporal patterns, which are crucial for accurately modeling node evolution. Third, among the three dynamic graph models (TGAT, TGN, and TREND), TGN shows the worst performance. This is due to the need to batch all edges within a snapshot to prevent repeated memory updates at the same timestamp, which limits its ability to capture long-term semantic changes. As a result, TGN is less effective at modeling intricate temporal dynamics compared to other continuous-time models.

From the perspective of **heterogeneity**, it is observed that the introduce of diverse semantic information helps to enhance the model's ability to capture the evolution patterns of graphs. Regarding static graph models, HGT tends to outperform GAT, showcasing the strengths of heterogeneous models in capturing complex relationships. In addition, in the realm of discrete-time models, both DyHART and HTGNN exhibit relatively well performance on the Aminer(D) dataset in the inductive setting and surpass other baseline models. This demonstrates the effectiveness of these approaches in leveraging heterogeneous semantics, allowing for a richer representation of the underlying graph structure.

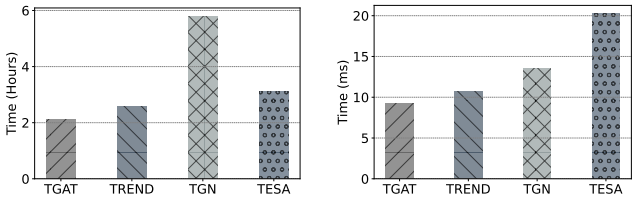
4.3 Micro Experiments

Ablation Study. Taking the Aminer(D) and Yelp(C) datasets as examples, Table 5 shows the results of our ablation study w.r.t. AUC. Specifically, *Ablation 1* removes the trajectory evolution module, *Ablation 2* excludes the semantic-aware aggregation module by ignoring edge types and performing simple neighbor message passing, and *Ablation 3* replaces the trajectory evolution module with an RNN-based time point process.

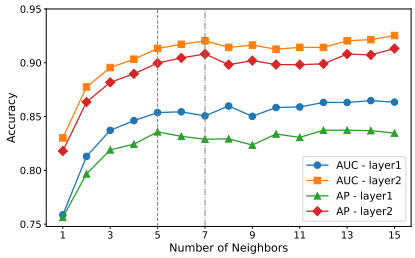
The results indicate that *removing the trajectory evolution module (Ablation 1)* leads to a performance drop, especially in the new-new prediction task. This highlights the importance of capturing the dynamic semantic changes in node behavior through trajectory modeling, as it helps in understanding the evolution of node interactions over time. Similarly, *excluding the semantic-aware aggregation module (Ablation 2)* results in further degradation, underscoring the necessity of considering different relation types in heterogeneous graphs to capture distinct semantic relationships. Finally, *replacing the attention-based trajectory evolution with an RNN (Ablation 3)* lowers the model performance, indicating that the attention mechanism more effectively captures the temporal dependencies and long-range semantic patterns compared to RNN.

Table 5: Ablation study on Aminer(D) and Yelp(C) datasets.

No.	Conf.	Aminer (D)		Yelp (C)	
		Trans	New-New	Trans	New-New
1	w/o tra_evo	0.9165	0.7490	0.7763	0.7544
2	w/o sem_agg	0.8743	0.7159	0.7729	0.7453
3	TeSa(RNN)	0.9133	0.7923	0.7803	0.7794
	TeSa(ATTN)	**0.9278**	**0.8155**	**0.8141**	**0.7908**



(a) Training time. **(b) Inference time.**



(c) Number of neighbors and layers.

Figure 3: (a) The training time of each epoch for the continuous-time models on Yelp(C); (b) The inference time for testing 1,000 edges. (c) Accuracy of TeSa when using different number of layers and neighbors;

Effect of Neighbor Number. In Figure 3c, we show the effect of the number of sampled neighbors on the AP and AUC metrics on Aminer(D) . We observe that the model's performance converges at a neighbor size of five for one-layer models (dashed line) and at seven for two-layer models (dash-dotted line). Notably, increasing the number of layers leads to a substantial enhancement in overall performance.

Model Efficiency. In Figure 3a and 3b, we present the training and inference time of **TeSa** alongside those of continuous-time baselines (TGAT, TREND and TGN) on Yelp(C). The results show that the training time of **TeSa** is close to TGAT and TREND, but significantly lower than TGN's. Unlike TGN, our model does not rely on a global memory module to store and track node semantic changes, and therefore is more efficient. Regarding inference time, **TeSa** requires 20.25 ms per 1,000 edges, which is approximately twice the time of TGAT's 9.29 ms. This additional time can be attributed to the necessity of trajectory construction and evolution, which is a trade-off for enhanced semantic representation. And further optimizing the efficiency of these evolution methods presents a valuable opportunity for future research.

5 Related Work

Dynamic Heterogeneous Graph Neural Networks. Dynamic Graph Neural Networks (DGNNs) have shown remarkable success in capturing the dynamic nature of temporal graphs. To extend these success to dynamic heterogeneous graphs(DHGs), a range of approaches have been proposed to capture the evolving dynamics and heterogeneous semantics within these graphs. Early works abstract DHGs as a sequence of heterogeneous snapshots. One category[8] encodes temporal features into edges and conducts heterogeneous message passing across the entire graph, which can lead to information leakage. Another category[3, 25, 30] focuses on heterogeneous message passing at each snapshot while utilizing sequence models to capture temporal variations. However, this discrete modeling has limitations in capturing continuous temporal dynamics and fine-grained temporal dependencies.

Recently, HPGE[10] has proposed a continuous modeling approach that represents DHGs as sequences of events. HPGE[10] effectively incorporates the Hawkes process into graph embedding to capture the excitation of various historical events on the current type-wise events. However, the Hawkes process is limited to modeling excitation effects, which constrain the ability to represent the complex and diverse interactions that might occur in such networks. In comparison, our method provides a continuous modeling framework that captures the entangled evolving dynamics and heterogeneous semantics in DHGs. By exploiting neural temporal point processes, our model can capture a wide range of complex interactions, without being confined to simply excitation effects.

Temporal Point Processes. Temporal point processes (TPPs) offer an effective mathematical framework for modeling event sequences in continuous time domains, which enables the representation of dynamic patterns inherent in sequences of events. Conventional TPPs, such as Poisson process[18] and Hawkes process[6], are constrained by their fixed mathematical forms, which significantly limit their expressive power. To overcome these limitations, researchers have merged the powerful representational capabilities of neural networks with TPPs, leading to the development of neural TPPs. Diverse RNN-based TPPs [2, 14, 29] leverage continuous state spaces and flexible transition functions, achieving better performance on many real-world datasets. Lately, a number of attention-based models have been suggested to capture sequences' long-range relationships, further improving the prediction performance. These attention-based neural TPPs [15, 33, 35] leverage the self-attention mechanism to effectively model dependencies across extended time spans, addressing the weaknesses of RNN-based approaches in handling such dependencies.

6 Conclusion

In this work, we address the problem of modeling continuous-time dynamic heterogeneous graphs, which capture evolving dynamics and heterogeneous semantics over time. We propose **TeSa**, a novel approach that leverages neural temporal point processes for capturing semantic changes entailed in node trajectory. This method excels in capturing intricate temporal dependencies and diverse node interactions, offering greater flexibility and expressiveness. Experimental results validate the effectiveness of our approach, highlighting its potential for various applications.

References

[1] Xu Da, Ruan Chuanwei, Korpeoglu Evren, Kumar Sushant, and Achan Kannan. 2020. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*.

[2] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1555–1564.

[3] Yujie Fan, Mingxuan Ju, Chuxu Zhang, and Yanfang Ye. 2022. Heterogeneous temporal graph neural network. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*. SIAM, 657–665.

[4] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of the web conference 2020*. 2331–2341.

[5] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[6] Alan G Hawkes. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika* 58, 1 (1971), 83–90.

[7] Ying He, Gongqing Wu, Desheng Cai, and Xuegang Hu. 2023. Meta-path based graph contrastive learning for micro-video recommendation. *Expert Systems with Applications* 222 (2023), 119713.

[8] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*. 2704–2710.

[9] Qiang Huang, Xin Wang, Susie Xi Rao, Zhichao Han, Zitao Zhang, Yongjun He, Quanqing Xu, Yang Zhao, Zhigao Zheng, and Jiawei Jiang. 2024. Benchtemp: A general benchmark for evaluating temporal graph neural networks. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4044–4057.

[10] Yugang Ji, Tianrui Jia, Yuan Fang, and Chuan Shi. 2021. Dynamic heterogeneous graph embedding via heterogeneous hawkes process. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*. Springer, 388–403.

[11] Renhe Jiang, Zhaonan Wang, Jiawei Yong, Puneet Jeph, Quanjun Chen, Yasumasa Kobayashi, Xuan Song, Shintaro Fukushima, and Toyotaro Suzumura. 2023. Spatio-temporal meta-graph learning for traffic forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 8078–8086.

[12] Wenjuan Luo, Han Zhang, Xiaodi Yang, Lin Bo, Xiaoqing Yang, Zang Li, Xiaohu Qie, and Jieping Ye. 2020. Dynamic heterogeneous graph neural network for real-time event prediction. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 3213–3223.

[13] Yuhong Luo and Pan Li. 2022. Neighborhood-aware scalable temporal network representation learning. In *Learning on Graphs Conference*. PMLR, 1–1.

[14] Hongyuan Mei, Guanghui Qin, Minjie Xu, and Jason Eisner. 2020. Neural Datalog through time: Informed temporal modeling via logical specification. In *International Conference on Machine Learning*. PMLR, 6808–6819.

[15] Hongyuan Mei, Chenghao Yang, and Jason Eisner. 2021. Transformer embeddings of irregularly spaced events and their participants. In *International conference on learning representations*.

[16] Hoang Nguyen, Radin Hamidi Rad, Fattane Zarrinkalam, and Ebrahim Bagheri. 2023. DyHNet: Learning dynamic heterogeneous network representations. *Information Sciences* 646 (2023), 119371.

[17] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[18] Siméon-Denis Poisson. 1837. *Recherches sur la probabilité des jugements en matière criminelle et en matière civile: précédées des règles générales du calcul des probabilités*. Bachelier.

[19] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.

[20] Guojiang Shen, Wenfeng Zhou, Wenyi Zhang, Nali Liu, Zhi Liu, and Xiangjie Kong. 2023. Bidirectional spatial–temporal traffic data imputation via graph attention recurrent neural network. *Neurocomputing* 531 (2023), 151–162.

[21] Chuan Shi, Xiao Wang, and S Yu Philip. 2022. *Heterogeneous graph representation learning and applications*. Springer.

[22] Georg Simmel. 1950. *The Sociology of Georg Simmel*. Free Press.

[23] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.

[24] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The world wide web conference*. 2022–2032.

[25] Xiao Wang, Yuanfu Lu, Chuan Shi, Ruijia Wang, Peng Cui, and Shuai Mou. 2020. Dynamic heterogeneous information network embedding with meta-path based proximity. *IEEE Transactions on Knowledge and Data Engineering* 34, 3 (2020), 1117–1132.

[26] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *International Conference on Learning Representations (ICLR)*.

[27] Qianlong Wen, Zhongyu Ouyang, Jianfei Zhang, Yiyue Qian, Yanfang Ye, and Chuxu Zhang. 2022. Disentangled dynamic heterogeneous graph learning for opioid overdose prediction. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2009–2019.

[28] Zhihao Wen and Yuan Fang. 2022. Trend: Temporal event and node dynamics for graph representation learning. In *Proceedings of the ACM Web Conference 2022*. 1159–1169.

[29] Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen Chu. 2017. Modeling the intensity function of point process via recurrent neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.

[30] Hansheng Xue, Luwei Yang, Wen Jiang, Yi Wei, Yi Hu, and Yu Lin. 2021. Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal rnn. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part I*. Springer, 282–298.

[31] Siqiao Xue, Xiaoming Shi, Zhixuan Chu, Yan Wang, Hongyan Hao, Fan Zhou, Caigao Jiang, Chen Pan, James Y. Zhang, Qingsong Wen, Jun Zhou, and Hongyuan Mei. 2024. EasyTPP: Towards Open Benchmarking Temporal Point Processes. In *International Conference on Learning Representations (ICLR)*.

[32] Chunyuan Yuan, Jiacheng Li, Wei Zhou, Yijun Lu, Xiaodan Zhang, and Songlin Hu. 2021. DyHGCN: A dynamic heterogeneous graph convolutional network to learn users' dynamic preferences for information diffusion prediction. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*. Springer, 347–363.

[33] Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. 2020. Self-attentive Hawkes process. In *International conference on machine learning*. PMLR, 11183–11193.

[34] Zeyang Zhang, Ziwei Zhang, Xin Wang, Yijian Qin, Zhou Qin, and Wenwu Zhu. 2023. Dynamic Heterogeneous Graph Attention Neural Architecture Search. *37th AAAI Conference on Artificial Intelligence* (2023).

[35] Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. 2020. Transformer hawkes process. In *International Conference on Learning Representations (ICLR)*.

A Datasets

Aminer(D) [10] is an academic citation dataset for papers published from 1990 to 2006. It contains three types of nodes (paper, author, and venue) and three types of relations: *paper-publish-venue*, *author-write-paper*, and *author-cooperate-author*. We separate time slices using the publication year, and this dataset is discrete in nature.

Yelp(D) is a business review dataset that includes user reviews and tips for businesses. Following [10], we focus on interactions within three categories of businesses: "American (New) Food," "Fast Food," and "Sushi," covering the period from January 2012 to December 2012. We separate time slices by month, and the graph includes the following types of relations: *user-review-business* and *user-tip-business*.

Yelp(C) [2] records activities on the platform from February 16, 2005, to January 19, 2022. We also construct two types of relations in the graph: *user-review-business* and *user-tip-business*. The temporal granularity is set at the minute level, capturing fine-grained temporal dynamics. This dataset is continuous in nature.

B Baselines

We evaluate our proposed model against seven baselines, which can be categorized based on the type of graphs they target: static homogeneous, static heterogeneous, dynamic homogeneous, and dynamic heterogeneous graphs. Below are brief descriptions of each baseline.

GAT [23] is a static homogeneous GNN that aggregates information from neighbors using an attention mechanism. It is designed for static graphs without considering node or edge types, and it serves as a representative baseline for static homogeneous methods.

HGT [8] is a static heterogeneous GNN that adopts mutual attention and parametrizes the attention mechanism differently for each node and relation type. It captures the structural heterogeneity of the graph but does not model temporal dynamics.

TGN [19] establishes a memory module to store node representations and uses an RNN to update the memory over time. It is designed for dynamic homogeneous graphs and operates in a continuous-time setting, focusing on temporal interactions without considering heterogeneous structures.

TGAT [1] utilizes self-attention to aggregate temporal-topological structures and learns the temporal evolution of node embeddings through continuous-time encoding. It targets dynamic homogeneous graphs and focuses on capturing the temporal evolution of nodes.

TREND [28] is based on a Hawkes process, which models the exciting effects between events. This dynamic homogeneous GNN captures temporal dependencies between events using the Hawkes process, but it does not handle heterogeneous graph structures.

DyHATR [30] is a dynamic heterogeneous GNN that employs hierarchical attention and temporal self-attention to capture both heterogeneous and temporal information in discrete-time dynamic graphs.

HTGNN [3] is a dynamic heterogeneous GNN that iteratively uses hierarchical attention and temporal self-attention to model complex dynamic interactions. It captures both dynamic and heterogeneous information in a discrete-time setting, making it well-suited for handling graphs with various node and relation types evolving over time.

[2] https://www.yelp.com/dataset

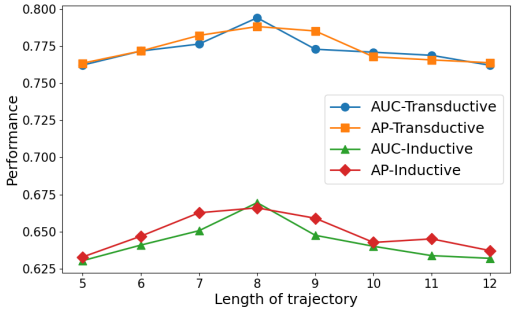


Figure 4: Performance variation with trajectory length on a dataset sampled from the first 1 million edges of Yelp(C).

Table 6: Model performance with and without the regularization term on the Aminer(D) and Yelp(D) datasets.

Dataset	Setting	w. Reg.	w.o. Reg.
Aminer(D)	Transductive	0.9278	0.9130
	New-Old	0.7985	0.7242
	New-New	0.8155	0.7662
Yelp(D)	Transductive	0.8276	0.8124
	New-Old	0.7065	0.7028
	New-New	0.7612	0.7186

C Additional experimental results

Length of Trajectory. As shown in Figure 4, the model's performance exhibits an initial increase followed by a decline as the trajectory length extends. This trend can be explained by the fact that longer trajectories allow the model to access more historical information, which helps capture the dynamic changes of nodes more effectively, thereby improving predictive accuracy. However, when the trajectory length becomes excessively long, it may encompass older events that are less relevant to the current prediction, potentially introducing noise and diminishing the model's decision-making capability. This observation suggests a trade-off in selecting the trajectory length, where an optimal length balances the coverage of useful historical information with the avoidance of irrelevant or noisy data.

Effect of Regularization Term. To assess the impact of the regularization term on the learned embedding of event type, we present the model performance w.r.t. AUC on the Aminer(D) and Yelp(D) datasets in Table 6. The results illustrate that incorporating the regularization improves the model's performance to a certain extent. For example, in the new-new setting of the Yelp(D) dataset, the AUC increases significantly from 0.7186 to 0.7612. The improvement in performance suggests that the regularization aids in

generating more meaningful embeddings, which in turn enhances the model's ability to generalize across different scenarios.

D Pseudocode

The overall training process of our model **TeSa**is illustrated in the algorithm shown below.

Algorithm 1: Overall Training Process of **TeSa**

Input: Dynamic heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{A}, \mathcal{R}, \mathcal{X})$, training event set $\mathcal{I}^{tr}$, batch size B

Output: Learned parameters Θ, including node embeddings and type-specific transformations

Initialization: Initialize node embeddings $\{\mathbf{h}_v\}_{v\in\mathcal{V}}$ and type-specific transformation matrices $\{M_{\phi(v)}\}$

while *not converged* **do**
- **for** *each batch of events* $\mathcal{B} \subset \mathcal{I}^{tr}$ *of size* B **do**
  - **for** *each event* $(u, v, t) \in \mathcal{B}$ **do**
    - $\mathbf{x}'_u \leftarrow M_{\phi(u)} \cdot \mathbf{x}_u$, $\mathbf{x}'_v \leftarrow M_{\phi(v)} \cdot \mathbf{x}_v$
    - $\mathbf{x}'_{(u,v)} \leftarrow M_{\phi((u,v))} \cdot \mathbf{x}_{(u,v)}$
    - $\mathcal{T}_u(t) \leftarrow$ `construct_trajectory`$(u, t,$ `event_history`$[u])$
    - $\mathcal{T}_v(t) \leftarrow$ `construct_trajectory`$(v, t,$ `event_history`$[v])$
    - $\mathbf{h}^{l,tr}_u(t) \leftarrow$ `trajectory_evolution`$(\mathcal{T}_u(t))$
    - $\mathbf{h}^{l,tr}_v(t) \leftarrow$ `trajectory_evolution`$(\mathcal{T}_v(t))$
    - **for** *each neighbor* $w \in \mathcal{N}(u) \cup \mathcal{N}(v)$ **do**
      - $\mathcal{T}_w(t) \leftarrow$ `construct_trajectory`$(w, t,$ `event_history`$[w])$
      - $\mathbf{h}^{l,tr}_w(t) \leftarrow$ `trajectory_evolution`$(\mathcal{T}_w(t))$
    - $\mathbf{h}^{l}_{u,R}(t) \leftarrow$ `semantic_aggregation`$(u, \{\mathbf{h}^{l,tr}_w(t) \mid w \in \mathcal{N}(u)\}, t)$
    - $\mathbf{h}^{l}_{v,R}(t) \leftarrow$ `semantic_aggregation`$(v, \{\mathbf{h}^{l,tr}_w(t) \mid w \in \mathcal{N}(v)\}, t)$
    - $\mathbf{h}^{l}_{u}(t) \leftarrow$ `combine`$(\mathbf{h}^{l,tr}_u(t), \mathbf{h}^{l}_{u,R}(t))$
    - $\mathbf{h}^{l}_{v}(t) \leftarrow$ `combine`$(\mathbf{h}^{l,tr}_v(t), \mathbf{h}^{l}_{v,R}(t))$
    - `node_features`$[u] \leftarrow \mathbf{h}^{l}_u(t)$
    - `node_features`$[v] \leftarrow \mathbf{h}^{l}_v(t)$
    - `event_history`$[u] \leftarrow$ `event_history`$[u] \cup \{(t, \Phi((u,v)), \mathbf{h}^{l}_u(t))\}$
    - `event_history`$[v] \leftarrow$ `event_history`$[v] \cup \{(t, \Phi((u,v)), \mathbf{h}^{l}_v(t))\}$
    - $\lambda_{u,v}(t) \leftarrow \mathtt{FCL}_r((\mathbf{h}^{L}_u(t) - \mathbf{h}^{L}_v(t))^2)$
    - $L_e \leftarrow -\log(\lambda_{u,v}(t)) - \log(1 - \lambda_{u,k}(t))$, where k is a negative sample
  - $\Theta \leftarrow \Theta - \eta\nabla_\Theta \left(\sum_{(u,v,t)\in\mathcal{B}} L_e + \eta_1\|\tilde{k}\|^2\right)$

return Θ

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009