Neural Temporal Walks: Motif-Aware Representation Learning on Continuous-Time Dynamic Graphs

Ming Jin
Monash University
ming.jin@monash.edu

Yuan-Fang Li
Monash University
yuanfang.li@monash.edu

Shirui Pan *
Griffith University
s.pan@griffith.edu.au

Abstract

Continuous-time dynamic graphs naturally abstract many real-world systems, such as social and transactional networks. While the research on continuous-time dynamic graph representation learning has made significant advances recently, neither graph topological properties nor temporal dependencies have been well-considered and explicitly modeled in capturing dynamic patterns. In this paper, we introduce a new approach, *Neural Temporal Walks* (`NeurTWs`), for representation learning on continuous-time dynamic graphs. By considering not only time constraints but also structural and tree traversal properties, our method conducts spatiotemporal-biased random walks to retrieve a set of representative motifs, enabling temporal nodes to be characterized effectively. With a component based on neural ordinary differential equations, the extracted motifs allow for irregularly-sampled temporal nodes to be embedded explicitly over multiple different interaction time intervals, enabling the effective capture of the underlying spatiotemporal dynamics. To enrich supervision signals, we further design a harder contrastive pretext task for model optimization. Our method demonstrates overwhelming superiority under both transductive and inductive settings on six real-world datasets [1].

1 Introduction

Continuous-time dynamic graphs (CTDGs) consist of temporal events with respect to nodes (e.g., node addition/deletion) and edges (i.e., temporal interactions), which naturally arise in many real-world systems such as social networks and knowledge graphs [13, 32]. Traditional studies in dynamic graph modeling manually extract expressive patterns that are beneficial for understanding the crucial laws behind [36]. For example, two people are likely to know each other if they have a common friend (Figure 1). Such a dynamic graph motif describes how social connections are established [31]. Other expressive patterns, such as feedforward control loops, have also been investigated [20]. However, manually extracting motifs is expensive, time-consuming, and requires domain knowledge, thus hindering the learning on dynamic graphs with more complicated laws [36].

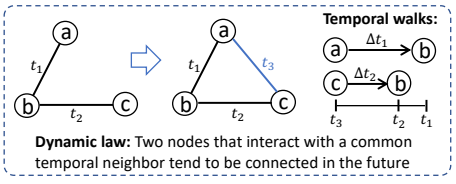


Figure 1: Temporal walks capture the law

The advent of graph neural networks (GNNs) makes it possible to understand more complicated graphs by automatically learning the laws behind [40, 42]. While GNNs have demonstrated great success in modeling static graphs, the research on dynamic graphs is still nascent. Current research on dynamic graph neural networks (DGNNs) faces two fundamental challenges. **Firstly**, the entangled spatial and temporal dependencies in real-world CTDGs typically need a specific design to model, preventing the direct use of

*Corresponding Author.

[1] Code is available at https://github.com/KimMeen/Neural-Temporal-Walks

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

off-the-shelf GNNs. To overcome this barrier, previous works simplify CTDGs to a series of static graph snapshots with *uniform time intervals* (i.e., discrete-time dynamic graphs, DTDGs) [24, 29]. However, this approximation compromises the modeling precision. Although [15] and [34] propose to learn on CTDGs directly, the inductiveness of the patterns they captured is not guaranteed because node identities are directly involved in their modeling process. Some recent methods [38, 27, 36] attempt to alleviate this issue. However, they only consider time but not topological and tree traversal properties when sampling temporal neighbors, limiting their ability to extract diverse and expressive patterns from dynamic systems. **Secondly**, temporal events in CTDGs occur irregularly (e.g., nodes a and c interact with b at different timestamps in Figure 1), resulting in a significant challenge in modeling temporal dependencies. Previous works typically bypass this challenge with the time encoding [38] to enable the use of message passing [38, 27] or sequence models [15, 36]. However, time encoding hurts model performance as temporal dependencies are modeled implicitly.

To tackle the above challenges, we propose the *Neural Temporal Walks* (`NeurTWs`) method for representation learning on CTDGs, which extracts and encodes informative dynamic graph motifs composed of irregularly-sampled temporal nodes. Specifically, motivated by [36], we propose *sptaiotemporal-biased random walks* to extract diverse and expressive patterns from a CTDG by not only considering time constrains but also topological properties and tree traversals, allowing the sampler to be better aware of the importance of temporal neighbors while maintaining the exploration and exploitation trade-off. To explicitly model temporal dependencies and capture essential dynamic laws more effectively, the extracted motifs consisting of irregularly-sampled temporal nodes are encoded with the proposed *continuous evolution* and *instantaneous activation* processes to learn time-aware node representations. The former process learns latent spatiotemporal dynamics across multiple interaction time intervals with an ordinary differential equation (ODE) function, and the second process regularizes the latent state trajectories with those irregularly-sampled observations.

On this basis, we acquire time-aware node embeddings by retrieving and encoding neighboring representative dynamic graph motifs, where a contrastive objective can be naturally designed to optimize `NeurTWs` by maximizing the mutual information between interacting temporal nodes. Compared with most existing works that use a simple link prediction objective, this harder contrastive pretext task helps enrich supervision signals, thus lifting the learning ability of `NeurTWs`.

On five benchmark datasets and a new dense e-commerce dataset, our method significantly and consistently outperforms all state-of-the-art methods in general. Specifically, it surpasses the strongest baselines by around 3% and 5% in all transductive and seven out of eight inductive link prediction tasks. It also achieves the best or on-par performance on dynamic node classification tasks. In summary, our technical contributions are three-fold: **(1)** We propose novel spatiotemporal-biased random walks to extract diverse and expressive patterns from CTDGs by considering not only time constraints but also topological and tree traversal properties; **(2)** We introduce a new perspective to encode dynamic graph motifs composed of irregularly-sampled temporal nodes, explicitly and better modeling the underlying spatiotemporal dynamics; **(3)** We integrate contrastive learning into dynamic graph modeling to enrich supervision signals, which lifts the learning ability of our model.

2 Related Work

Dynamic Graph Neural Networks (DGNNs). Existing DGNNs can be broadly classified into two categories based on their inputs. Discrete-time DGNNs operate over a sequence of evenly-sampled static graph snapshots, where different strategies are proposed to model spatial and temporal clues, e.g., combining GNNs with sequence models [30, 24, 5, 29]. Our work relates to the second category, continuous-time DGNNs, where time-dependent node or edge embeddings are learned directly on CTDGs. Among these works, an in-demand design updates latent node states by aggregating k-hop neighborhood information with temporal message passing. For instance, TGAT [38] samples a set of k-hop temporal neighbors and proposes learnable time encodings to preserve time information in message passing. TGN [27] further equips TGAT with a node memory update mechanism as in [15]. Another line of research leverages random walks to learn on CTDGs. Specifically, CTDNE [23] is the first to propose a CTDG embedding method with temporal walks. CAWs [36] extend this concept with anonymous temporal walks and uses a recurrent net to learn walk embeddings that are further aggregated when calculating interactive representations. Our method is different from prior walk-based approaches in three aspects: (1) We propose a new perspective on sampling temporal walks. While prior arts only consider time constraints, our method leverages multidimensional information,

allowing the model to explore diverse and expressive patterns from CTDGs; (2) We propose a novel and intuitive motif embedding method to model latent spatiotemporal dynamics among irregularly-sampled temporal nodes on CTDGs without relying on time encodings, which allows temporal dependencies to be modeled explicitly; (3) We replace the simple link prediction-based learning objective with a more challenging contrastive pretext task, which helps provide stronger supervision signals.

Neural Ordinary Differential Equations (NODEs). Chen *et al.* [2] propose a new paradigm that generalizes discrete deep neural networks by parameterizing the derivative of latent states. This concept has been applied in research areas including time series forecasting [12, 28] and computer vision [9, 1]. Recently, some works have extended NODEs to the graph domain, where most consider building deeper GNNs while alleviating the negative impacts of over-smoothing [37, 25]. Notably, the time information is absent among those works. In dynamic graph learning, most ODE-based works focus on discrete-time settings [12, 6, 8], and only a few extend NODEs to learn on CTDGs [7]. In this paper, inspired by the research on time series forecasting [28], we propose `NeurTWs` to encode extracted motifs with irregularly-sampled temporal nodes on CTDGs, which is fundamentally different from [7]: We directly integrate over multiple interaction time intervals to explicitly model the latent spatiotemporal dynamics across different temporal nodes with an ODE function, while [7] relies on a time encoding-assisted message passer to learn from historical temporal events.

Graph Contrastive Learning (GCL). Recently, GCL has achieved great success in graph self-supervised learning [19]. Most existing works are on static graphs [35, 43, 44, 11]. While some studies have explored the possibility of dynamic graph contrastive learning, many of them are on DTDGs. For example, STGCL [18] enhances the model's forecasting ability with DTDG augmentations and an auxiliary contrastive loss. A similar design also exists in [41, 39] and [26]. On CTDGs, Tian *et al.* [33] propose to maximize the agreement between time-aware node embeddings at different time points. In DySubC [10], the mutual information between a node and its surrounding temporal subgraphs is maximized. Different from these works, we design an effective pretext task for model optimization, where the mutual information between two nodes in an interaction is maximized. Meanwhile, we push nodes away in the embedding space if there are no temporal interactions.

3 Problem Formulation

We start by formally introducing the learning problem on CTDGs. A complete notation table is in Appendix A. This paper defines a CTDG as a stream of temporal interactions, i.e., $\mathcal{G} = \{(e_i, t_i)\}_{i=1}^{N}$, where each interaction has two nodes at a specific time, e.g., $(e_i, t_i) \coloneqq (\{u_i, v_i\}, t_i),\ t_i \in \mathbb{R}^+$. As many real-world CTDG datasets are unattributed and for simplicity, we first assume these temporal interactions are without node and edge attributes and later we will discuss how our method is extended to learn on attributed CTDGs. Facing the challenge of lacking label information, DGNNs are typically supervised by temporal interactions [45]. Thus, dynamic link prediction is a widely adopted testbed to evaluate how accurate a DGNN is in predicting future interactions with the observation of historical events. Specifically, given two nodes u and v at time t in $\mathcal{G}$, we aim to learn their time-aware embeddings $\overline{h}_u$ and $\overline{h}_v$, where the presence of an interaction between them can be predicted with a downstream classifier, i.e., $\widehat{y}_{u,v,t} = \text{clf}(\overline{h}_u, \overline{h}_v)$. The ground truth $y_{u,v,t} = 1$ if there exists an interaction between u and v at time t otherwise $y_{u,v,t} = 0$. On this basis and with learned time-aware node representations, conducting other downstream tasks, such as dynamic node classification with another classifier, is also feasible.

4 The Proposed Method: Neural Temporal Walks

4.1 Preliminaries: Temporal Walks and Dynamic Graph Motifs

Given a dynamic graph $\mathcal{G}$, we define a motif as a subset of temporal nodes with their interactions within a defined time range [14], e.g., $0 \leq t \leq q$. As those motifs reflect certain dynamic laws in a CTDG, it is desirable to characterize a temporal node with its surrounding motifs.

Definition 4.1.1 (Temporal Walk). Given a dynamic graph $\mathcal{G}$, we denote the interactions that are directly associated with a node u before a cut time t as $\mathcal{G}_{u,t} = \{(e, t') \mid t' < t,\ u \in e\ ,\ (e, t') \in \mathcal{G}\}$. A (time-reversed) temporal walk rooted at node u at time t is defined as W, which is a sequence of

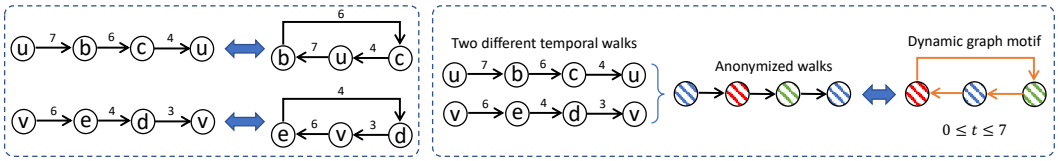
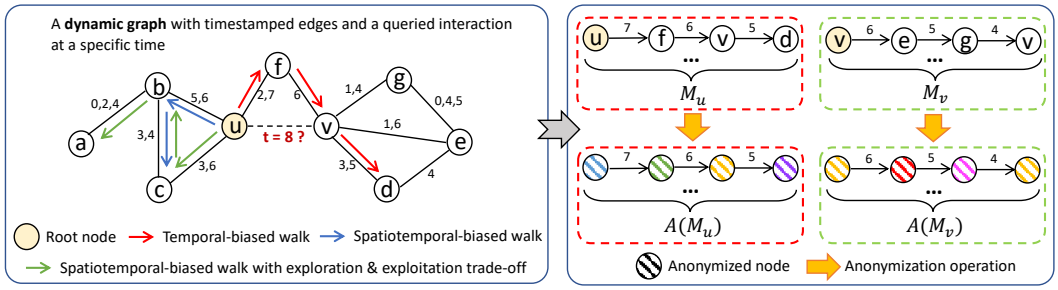


Figure 2: Triadic closures and the dynamic graph motifs: Two example temporal walks form two different triadic closures but represent the same motif within the time range $0 \leq t \leq 7$.



(a) Dynamic Graph Motif Extraction (b) Walk Anonymization

Figure 3: Temporal walk sampling and anonymization. Given a CTDG and a queried interaction $(u, v, 8)$, we first extract surrounding temporal motifs by sampling a set of diverse and expressive temporal walks started from u and v, respectively, denoted as M_u and M_v. Then, the walks in two sets are anonymized by replacing nodes' identities with their position encodings.

temporal nodes as in [36], i.e., node w_i at time t_i with $w_0 := u$ and $t_0 := t$:

$$W = \{(w_i, t_i) \mid i \in \mathbb{N},\ 0 \leq i \leq l,\ t_0 > t_1 > \cdots > t_l,\ (\{w_i, w_{i-1}\}, t_i) \in \mathcal{G}_{w_{i-1}, t_{i-1}} \text{ for } i \geq 1\}, \tag{1}$$

where l is the walk length. We also use $W[i][0]$ and $W[i][1]$ (i.e., w_i and t_i in (w_i, t_i) respectively) to denote the specific node and time in the i-th step.

Each walk W rooted at a temporal node can actually be regarded as one of its surrounding motifs if $t_1 - t_l$ is bounded within the defined motif time range. A concrete example is given in Figure 2, where two walks in the leftmost side form two triadic closures, which essentially represent the same motif on the rightmost side after an individual walk anonymization [21]. The necessity of anonymization is to replace original node identities in walks with their relative identities, which maps each walk to a particular pattern and thus connects temporal walks and dynamic graph motifs.

Definition 4.1.2 (Anonymous Walk). Given a temporal node w and a walk W, the anonymization operation $A(\cdot)$ is defined as follows [36]:

$$A(w; W) = |\{v_0, \cdots, v_{i^*} \mid v_i \in W\}|, \text{where } i^* \text{ is the smallest index s.t. } v_{i^*} = w. \tag{2}$$

4.2 Dynamic Graph Motif Extraction

Temporal Walk Sampling. Existing path-based methods mainly employ a temporal-biased sampling method when extracting dynamic graph motifs [23, 36]. Specifically, given a node u at time t, the probability of its neighbor a in $(\{a, u\}, t') \in \mathcal{G}_{u,t}$ to be sampled is proportion to $exp(\alpha(t' - t))$, which discounts stale neighbors and tends to sample more current nodes with timestamps closer to t. A larger α emphasizes more on this bias. Although more current neighbors are more likely to be informative, the underlying topological and tree traversal properties are not respected, which hinders the extraction of diverse and expressive patterns. Here, we propose *spatiotemporal-biased random walks* with the exploitation and exploration trade-off.

Our motivations are twofold: (1) Most-recent neighbors should be allocated a larger sampling probability $\propto exp(\alpha(t' - t))$ as they are typically more informative w.r.t. a root node at time t. In Figure 3(a), given a root node u and its two temporal neighbors f and c, a temporal-biased sampling path is more likely to be $u \to f$ instead of $u \to c$; (2) Neighbors with higher connectivity need to be emphasized to allow exploring more diverse and potentially expressive motifs. Given a node a at time

t', we use its degree $d_a = |\mathcal{G}_{a,t'}|$ to reify its connectivity, thus the proposed spatial-biased probability $\propto exp(-\beta/d_a)$ with a hyperparameter β to control the bias intensity. Algorithm 1 illustrates the walk sampling procedures with the above considerations. Given a node u at time t, the probability of its temporal neighbor a to be sampled is the average of the following normalized probabilities:

$$Pr_t(a) = \frac{exp(\alpha(t_a - t))}{\sum_{a' \in \mathcal{G}_{u,t}} exp(\alpha(t_{a'} - t))} \quad (3) \qquad Pr_s(a) = \frac{exp(-\beta/d_a)}{\sum_{a' \in \mathcal{G}_{u,t}} exp(-\beta/d_{a'})} \quad (4)$$

Algorithm 1 Sampling Temporal Walks

Require: Root node w_0, cut time t_0, $\mathcal{G}$, C, l
Initialize $\{W_c = \big((w_0, t_0)\big) \mid 1 \leq c \leq C\}$
for i in $1, 2, \cdots, l$ **do**
  for j in $1, 2, \cdots, C$ **do**
    $w_p, t_p \coloneqq W_j[i][0], W_j[i][1]$
    Initialize $d_w = 0$ for all $w \in \mathcal{G}_{w_p,t_p}$
    for (e, t) in $\mathcal{G}_{w_p,t_p}$ **do**
      Let $e \coloneqq \{w, w_p\}$, $d_w = |\mathcal{G}_{w,t}|$
    end for
    Sample one $(e, t) \in \mathcal{G}_{w_p,t_p}$ with prob. $\propto exp(\alpha(t - t_p))$ and $exp(-\beta/d_w)$
    Let $e \coloneqq \{w, w_p\}$, $W_c = W_c||(w, t)$
  end for
end for
return $\{W_c \mid 1 \leq c \leq C\}$

Although Algorithm 1 complements the temporal-biased schema by considering the additional topological information, it may overly encourage the depth-first search (DFS), which could misleadingly sample many homogeneous motifs with a limited budget. Take an extreme example, if $|M_u|$ is restricted to 3 in Figure 3(a), paths $u \to b \to c \to u$ may be sampled three times, leaving no room to explore $u \to c \to b \to a$ and $u \to f \to v \to d$. Thus, we design an exploitation & exploration trade-off to regularize the walk sampling with another probability:

$$Pr_e(a) = \frac{exp(-\gamma s_a)}{\sum_{a' \in \mathcal{G}_{u,t}} exp(-\gamma s_{a'})}, \quad (5)$$

where s_a and γ denotes the traversal times of node a and the intensity of such a regularization.

Our complete walk sampling algorithm and its complexity analysis are in Appendices B.1 and B.3. In a nutshell, given a temporal node, the probability of its neighbor to be sampled is the average of the probabilities defined in Equations 3, 4 and 5. In `NeurTWs`, given a queried interaction between two temporal nodes u and v as shown in Figure 3, we sample a set of C temporal walks rooted at each node with length l, denoted as M_u and M_v.

Anonymization. Walk anonymization replaces node identities with position encodings (aka relative identities), which injects structural information while maintaining the inductiveness of `NeurTWs`. A drawback of Equation 2 is that the position encoding of each node only depends on its specific walk, leading anonymous walks rooted at the same node sharing different name spaces [36]. Thus, we consider two practical solutions: *unitary* and *binary anonymization* to address this problem. For a temporal node w in at least one walk rooted at node u, its unitary anonmization w.r.t. u considers the name space defined over M_u, the set of walks rooted at u:

$$A(w; M_u)[i] = |\{W \mid w = W[i][0],\ W \in M_u\}|, \text{where } i \in \{0, \cdots, l\}. \quad (6)$$

In Equation 6, the identity of w is replaced by a vector $A(w; M_u)$ with length l, where the i-th element counts the number of walks that have node w appearing in position i.

While unitary anonymization anonymizes node w w.r.t. M_u as $A(w; M_u)$, for interacting root nodes u and v, $A(w; M_u)$ and $A(w; M_v)$ belong to different name spaces. Since DGNNs are typically supervised by temporal interactions, establishing the correlations between $W \in M_u \cup M_v$ may be beneficial. Thus, the binary anonymization is defined as follows [36]:

$$A(w; M_u, M_v) = A(w; M_u) \mid\mid A(w; M_v), \quad (7)$$

where $||$ denotes the concatenation operation to establish the connections among motifs in M_u and M_v. In the rest of the paper, we abbreviate the two anonymization strategies as $A(w)$ for simplicity and denote $\widehat{W} = \{\big(A(w_i), t_i\big) \mid (w_i, t_i) \in W \text{ for } i = 0, 1, \cdots, l\}$ as an anonymous walk.

4.3 Neural Temporal Walk Encoding

A significant challenge to model CTDGs is that interactions occur irregularly. Previous works bypass this challenge by concatenating node attributes with extra time encodings when aggregating the neighbourhood information [38, 27, 36], where temporal dependencies are modeled implicitly. A detailed discussion is in Appendix D.2. To encode a motif with irregularly-sampled temporal nodes, we explicitly integrate over multiple interaction time intervals to learn the latent spatiotemporal dynamics with those discrete observations. Figure 4 and Algorithm 2 describe our method in a nutshell, which consists of two interleaving steps: *Continuous evolution* and *instantaneous activation*.

Algorithm 2 Neural Temporal Walk Encoding

Require: An anonymous temporal walk $\widehat{W} = \{(A(w_i), t_i) \mid (w_i, t_i) \in W \text{ for } i = 0, 1, \cdots, l\}$

1: Reverse the order of elements in $\widehat{W}$
2: $t_{-1} = t_0$, $h_{-1} = \mathbf{0}$
3: **for** i in $0, 1, 2, \cdots, l$ **do**
4: $\quad h_i^{'} = \text{ODESolve}(h_{i-1}, f_\theta, t_{i-1}, t_i)$
5: $\quad A^{'}(w_i) = \text{MLP}_\psi(A(w_i))$
6: $\quad h_i = g_\phi(h_i^{'}, A^{'}(w_i))$
7: **end for**
8: **return** The walk embedding h_l

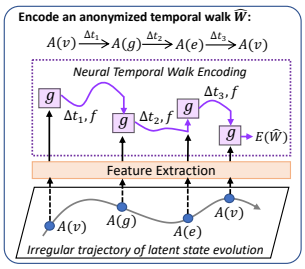


Figure 4: The spatiotemporal dynamics behind irregularly-sampled temporal nodes are explicitly modeled.

Continuous Evolution. Given a series of temporal nodes at different time, i.e., $(A(w_i), t_i) \in \widehat{W}$ and ensuring $t_{i-1} < t_i$ by reversing the order of elements in $\widehat{W}$, the latent spatiotemporal dynamics among those nodes are modeled as follows:

$$h_i^{'} = h_{i-1} + \int_{t_{i-1}}^{t_i} f(h_t, \theta)\, dt, \tag{8}$$

where h_{i-1} denotes the latent states after encoding $(A(w_{i-1}), t_{i-1}) \in \widehat{W}$. We define the ODE function $f(h_t, \theta)$ as an autoregressive gated recurrent unit parameterized by θ. See Appendix B.2.

Instantaneous Activation. The latent state evolution in Equation 8 conditions on a series of discrete observations. Thus, we define a function to activate latent state trajectories with instantaneous inputs:

$$h_i = g(h_i^{'}, A^{'}(w_i), \phi), \tag{9}$$

where $g(\cdot, \phi)$ can be a standard RNN cell parameterized by ϕ, and $A^{'}(w_i) = \text{MLP}(A(w_i), \psi)$ denotes the linear mapping of a discrete observation $A(w_i)$ in an anonymous walk $\widehat{W}$.

Compared with [36], we naturally model spatiotemporal dynamics behind walks with irregularly-sampled temporal nodes, where the time information has been explicitly reflected in this modeling process. On this basis, for a temporal node u, we obtain its time-aware representation by pooling the embedding of walks in M_u, denoted as $\overline{h}_u$. In this paper, we adopt the sum pooling for simplicity.

4.4 Contrastive Optimization

A self-supervised pretext task is required to train DGNNs due to the scarcity of the labeling information. Most current works formulate their learning problems as a binary classification task, where the existence of an interaction is predicted. From the contrastive point of view, a binary cross-entropy loss essentially forms a particular case of the Jensen-Shannon estimator [19], where the number of negatives is one and thus provides limited supervision signals.

Here, we introduce a harder contrastive pretext task, where the mutual information between interacting temporal nodes (e.g., node u and v in Figure 3) is maximized. We detail the complete training algorithm of our method in Appendix B.4. In brief, the following noise-contrastive loss is minimized in the proposed approach:

$$\mathcal{L} = -\mathbb{E}\Big[\log \frac{exp\big(\text{sim}(\overline{h}_u, \overline{h}_v)\big)}{exp\big(\text{sim}(\overline{h}_u, \overline{h}_v)\big) + \sum_{v' \in \mathcal{G}, v' \neq v} exp\big(\text{sim}(\overline{h}_u, \overline{h}_{v'})\big)}\Big]. \tag{10}$$

$\text{sim}(\cdot)$ measures the similarity between two entities, i.e., $\text{sim}(\overline{h}_u, \overline{h}_v) = \sigma\big(\text{MLP}(\overline{h}_u, \overline{h}_a, \xi)\big)$, where $\sigma(\cdot)$ and ξ are sigmoid activation and trainable parameters.

4.5 Extension and Discussion

Learning on Attributed CTDGs. Our method can be easily extended to model CTDGs with node and interaction attributes. To achieve this, we only need to slightly modify Equation 9 as follows:

$$h_i = g(h_i^{'},\ A^{'}(w_i) \,||\, X_{w_i} \,||\, X_{e_i},\ \phi), \tag{11}$$

Table 1: The dataset statistics. Average interaction intensity $\lambda = 2N/(|V|T)$ [16, 36] embodies the density of interactions in a fixed period, where T and $|V|$ are dataset duration and number of nodes.

Statistics	CollegeMsg	Enron	Taobao	MOOC	Wikipedia	Reddit
# Nodes & Interactions	1,899 & 59,835	143 & 62,617	64,703 & 77,436	7,144 & 411,749	9,227 & 157,474	10,984 & 672,447
Duration (second)	16,621,303	72,932,520	36,000	2,572,086	2,678,373	2,678,390
# Nodes & Interaction attributes	0 & 0	0 & 0	0 & 4	0 & 4	172 & 172	172 & 172
Average interaction intensity λ	3.79×10^{-6}	1.2×10^{-5}	6.64×10^{-5}	4.48×10^{-5}	1.27×10^{-5}	4.57×10^{-5}
# Dynamically labeled nodes	-	-	-	-	217	366

where X_{w_i} and X_{e_i} are linearly mapped features of node w_i and edge $e_i := \{w_{i-1}, w_i\}$ in $\widehat{W}$.

Batching and Computational Complexity. One computational challenge is that each walk contains irregularly-sampled nodes at different timestamps, requiring the model to separately solve C different ODEs to calculate the embedding of a root node, thus hindering training in batches. To solve all ODEs at once in a minibatch with B sampled interactions, we unify the integral time among all ODEs (see Appendix B.3) to a certain range, which results in a lower time complexity of $\mathcal{O}(l)$ instead of $\mathcal{O}(Bl)$. In Appendix B.3, we also provide detailed complexity analysis and discuss how to make solving Equation 8 tractable with very large time intervals.

5 Experiments

5.1 Experimental Setting

Baselines. Our model is compared with six state-of-the-art methods in modeling CTDGs. They can be divided into two categories based on their intrinsic mechanisms: (1) message passing-based methods, including DyRep [34], TGAT [38], and TGN [27]; (2) sequential model-based methods, including CTDNE [23], JODIE [15], and CAWs [36]. More details can be found in Appendix C.2.

Datasets. We evaluate model performance on six real-world datasets. CollegeMsg [17] is a social network dataset consisting of message sending activities. Enron [17] is an email communication network. Taobao [46] is an attributed user behavior dataset containing user-item interactions. MOOC [17] is an attributed network consisting of student-course interactions. Wikipedia and Reddit [15] are two bipartite interaction networks consisting of editor-page and user-post activities with rich attributive information. The dataset statistics are summarized in Table 1, and their detailed descriptions are in Appendix C.1.

Evaluation Protocols. For link prediction, we follow the evaluation protocols of CAWs [36] and consider two different downstream tasks for evaluation: *transductive* and *inductive* link prediction.

In *transductive* link prediction, we sort and divide all N interactions in a dataset by time into three separate sets for training, validation, and testing. Specifically, the ranges of training, validation, and testing sets are $[0, N_{trn})$, $[N_{trn}, N_{val})$, $[N_{val}, N]$, where N_{trn}/N and N_{val}/N are 0.7 and 0.85.

In *inductive* link prediction, we use the same splits but mask a proportion of nodes (10%) and associated interactions during training, which are predicted during inference to evaluate the model inductiveness. Specifically, we first remove all interactions connected with masked nodes in the training set and then remove all interactions *not* associated with them in the validation and testing sets. In particular, we have two specific settings: (1) *New-Old* tasks require the model to predict the interactions with one unobserved (i.e., masked) and one observed nodes; and (2) *New-New* tasks aim to predict the interactions between all unobserved nodes.

For dynamic node classification, we follow the evaluation protocol in [27]. Specifically, we first obtain a model under the setting of *transductive* link prediction. Then, we train and test a separate classifier on top of this pre-trained model with the temporal nodes observed in $[0, N_{val})$ and $[N_{val}, N]$.

Training Details. We implement and train all models under a unified evaluation framework with the Adam optimizer. The tuning of primary hyperparameters is discussed in Appendix C.3. In solving ODEs, we use the Runge-Kutta method with the number of function evaluations set to 8 by default. For fair comparisons and simplicity, we use sum-pooling when calculating node representations in both our method and CAWs. We also test `NeurTWs`†, which is equipped with the binary anonymization, while `NeurTWs` adopts the default unitary anonymization. All methods are tuned thoroughly with nonlinear 2-layer and 3-layer perceptrons to conduct downstream link

Table 2: Transductive and inductive link prediction performances w.r.t. AUC. We use **bold font** and underline to highlight the best and second best performances. `NeurTWs†` is a vairant of our method with the binary anonymization.

Task		Method	CollegeMsg	Enron	Taobao	MOOC
Transductive		JODIE [15]	0.5846 ± 0.038	0.8714 ± 0.011	0.8477 ± 0.015	0.6815 ± 0.014
		DyRep [34]	0.5297 ± 0.042	0.8632 ± 0.013	0.8462 ± 0.012	0.6195 ± 0.018
		TGAT [38]	0.7528 ± 0.004	0.6592 ± 0.012	0.5400 ± 0.005	0.6750 ± 0.035
		TGN [27]	0.8990 ± 0.003	0.8944 ± 0.015	0.8484 ± 0.029	0.7703 ± 0.032
		CAWs [36]	0.9002 ± 0.002	0.9520 ± 0.002	0.8719 ± 0.001	0.6948 ± 0.053
		`NeurTWs`	0.9526 ± 0.002	0.9564 ± 0.005	$\mathbf{0.9100 \pm 0.014}$	$\mathbf{0.7756 \pm 0.031}$
		`NeurTWs†`	$\mathbf{0.9750 \pm 0.004}$	$\mathbf{0.9704 \pm 0.012}$	0.8911 ± 0.014	0.7470 ± 0.028
Inductive	New-Old	JODIE [15]	0.4589 ± 0.028	0.8182 ± 0.022	0.7626 ± 0.002	0.6304 ± 0.006
		DyRep [34]	0.4486 ± 0.021	0.7241 ± 0.025	0.7641 ± 0.012	0.5504 ± 0.010
		TGAT [38]	0.7240 ± 0.008	0.6131 ± 0.049	0.5537 ± 0.018	0.6410 ± 0.024
		TGN [27]	0.8699 ± 0.007	0.7068 ± 0.116	0.8706 ± 0.008	0.6968 ± 0.008
		CAWs [36]	0.8911 ± 0.015	$\mathbf{0.9612 \pm 0.002}$	0.8744 ± 0.004	0.7479 ± 0.023
		`NeurTWs`	0.9575 ± 0.011	0.9525 ± 0.002	$\mathbf{0.9316 \pm 0.018}$	$\mathbf{0.7822 \pm 0.004}$
		`NeurTWs†`	$\mathbf{0.9699 \pm 0.010}$	0.9566 ± 0.007	0.9037 ± 0.013	0.7772 ± 0.006
	New-New	JODIE [15]	0.5135 ± 0.048	0.7537 ± 0.025	0.7791 ± 0.004	0.8243 ± 0.007
		DyRep [34]	0.5813 ± 0.066	0.7184 ± 0.061	0.7716 ± 0.017	0.5288 ± 0.021
		TGAT [38]	0.7283 ± 0.029	0.6340 ± 0.032	0.5479 ± 0.025	0.6365 ± 0.014
		TGN [27]	0.7745 ± 0.102	0.9217 ± 0.026	0.8701 ± 0.011	0.6448 ± 0.053
		CAWs [36]	0.8974 ± 0.009	0.9777 ± 0.001	0.8762 ± 0.004	0.7558 ± 0.036
		`NeurTWs`	0.9649 ± 0.008	$\mathbf{0.9906 \pm 0.007}$	$\mathbf{0.9242 \pm 0.005}$	$\mathbf{0.8329 \pm 0.010}$
		`NeurTWs†`	$\mathbf{0.9768 \pm 0.008}$	0.9858 ± 0.015	0.9140 ± 0.013	0.8302 ± 0.007

prediction and node classification tasks, and we adopt the commonly used Area Under the ROC Curve (AUC) and Average Precision (AP) as the evaluation metrics. More implementation details can be found in Appendix C.3.

5.2 Results and Discussion

The link prediction performance of our method and state-of-the-art baselines are summarized in Table 2 (w.r.t. AUC) and Appendix D.1 (w.r.t. AP). As can be seen from this table, our method achieves the best performance on different tasks and datasets in general. Notably, `NeurTWs` surpasses the strongest model, CAWs, significantly on both unattributed and attributed CTDGs, e.g., 7.76% and 4.78% on average on CollegeMsg and Taobao. Particularly, `NeurTWs` and `NeurTWs†` achieve a strong AUC result ≥ 0.95 on CollegeMsg, while all baselines have their AUCs ≤ 0.9, demonstrating the effectiveness of the proposed method.

Table 3: Dynamic node classification performance w.r.t. AUC. We use **bold font** and underline to highlight the best and second best performances. The baseline results are taken from [27].

Method	Wikipedia	Reddit
CTDNE [23]	0.7589 ± 0.005	0.5943 ± 0.006
JODIE [15]	0.8484 ± 0.012	0.6183 ± 0.027
DyRep [34]	0.8459 ± 0.022	0.6291 ± 0.024
TGAT [38]	0.8369 ± 0.007	0.6556 ± 0.007
TGN [27]	0.8781 ± 0.003	$\mathbf{0.6706 \pm 0.009}$
`NeurTWs`	$\mathbf{0.8851 \pm 0.003}$	0.6652 ± 0.022

Specifically, on both transductive and inductive tasks, we make the following observations. (1) Our method performs well on both attributed and unattributed datasets, while the effectiveness of baseline models varies significantly. For instance, JODIE and DyRep have enormous performance gaps between Taobao and CollegeMsg in all three tasks. It indicates that our `NeurTWs` method is better at capturing essential dynamic laws. This superiority can be attributed to our spatiotemporal walks and associated encoding mechanism; (2) Our method is effective under both transductive and inductive settings. On the contrary, some baseline models cannot well generalize to predict interactions with unseen nodes (e.g., the performance of JODIE and DyRep drops on inductive tasks) because they rely on node identities. TGAT, TGN, and CAWs can yield better performances but leave room to improve. It is worth noting that simple unitary anonymization (i.e., `NeurTWs`) and more complex binary anonymization (i.e., `NeurTWs†`) yield similar performance in most cases. Our method with the default, simpler unitary anonymization performs significantly better than CAWs, which utilizes binary anonymization, again demonstrating the superiority of our approach in modeling

Table 4: Ablation study with the proposed NeurTWs method. The performance in predicting *all inductive* interactions is reported.

No.	Configuration	CollegeMsg		Taobao	
		AUC	AP	AUC	AP
0	Full model (NeurTWs)	**0.958 ± 0.01**	**0.966 ± 0.01**	**0.938 ± 0.02**	**0.933 ± 0.02**
1	w/o T-biased probability	0.918 ± 0.02	0.928 ± 0.02	0.932 ± 0.03	0.927 ± 0.01
2	w/o S-biased probability	0.949 ± 0.02	0.958 ± 0.02	0.915 ± 0.01	0.915 ± 0.01
3	w/o E&E-biased probability	0.957 ± 0.01	0.965 ± 0.01	0.926 ± 0.01	0.927 ± 0.01
4	w/o continuous evolution	0.868 ± 0.02	0.898 ± 0.01	0.860 ± 0.05	0.901 ± 0.02
5	w/o contrasitve learning	0.954 ± 0.01	0.962 ± 0.01	0.935 ± 0.01	0.932 ± 0.01

CTDGs without relying on sophisticated position encodings; (3) Comparing the results on Taobao, our method's improvements over CAWs are more significant than on CollegeMsg, e.g., 7.48% vs. 3.81% on transductive and 7.91% vs. 5.26% on inductive settings. Similar observations can also be found when compared on MOOC and CollegeMsg. This is because the CollegeMsg dataset has a lower average interaction intensity (i.e., $\lambda = 3.79 \times 10^{-6}$) compared with Taobao and MOOC datasets (i.e., $\lambda = 6.64 \times 10^{-5}$ and 4.48×10^{-5}). A lower λ indicates a larger average time span in a temporal walk, for which our proposed continuous evolution process is more effective.

In addition, we compare the performance of our method (i.e., NeurTWs) on dynamic node classification with state-of-the-art baselines as shown in Table 3, where our approach achieves the best or on-par performances, further confirming the effectiveness of the proposal. We do not report the results of NeurTWs† because node classification mainly considers properties of temporal nodes rather than interactions.

5.3 Ablation Study

In Table 4, we report the results of our ablation study on predicting *inductive* interactions in both new-old and new-new settings. Specifically, ablations 1, 2, and 3 remove Equations 3, 4, and 5 when sampling temporal walks. Ablation 4 disables Equation 8 when encoding an anonymous walk. Ablation 5 replaces our contrastive loss with a binary cross entropy loss. From the above variants, we see performances degradation when retrieving motifs without considering the temporal or spatial information, demonstrating the effectiveness of the proposed spatiotemporal-biased walk sampler. Moreover, disabling the exploitation & exploration trade-off also hurts performances. When removing the continuous evolution, more severe performance drops are found on CollegeMsg, further demonstrating that NeurTWs excel on CTDGs with more sparse interactions where existing works usually fail to model them well. The performance drops also exist when our contrastive pretext task is disabled. A specific study related to continuous evolution is in Appendix D.2.

5.4 Parametric Sensitivity

We study the important settings in NeurTWs and have the following observations: (1) For each dataset, there are sweet spots in balancing three intensities (i.e., α, β, and γ). Specifically, different datasets with different average interaction intensities, namely λ, prefer different temporal-biased intensities. For example, a relatively large α benefits on CollegeMsg but hurts the performance on Taobao as shown in Figure 5(a). On both datasets, overly emphasizing the topology information with a large β hurts the performance (see Figure 5(b)) because the temporal information is overly neglected. However, setting a relatively large γ seems beneficial (see Figure 5(c)), indicating the importance of balancing exploitation and exploration in temporal walk sampling; (2) We also investigate the choice of walk length l and number of walks C as shown in Figures 5(e) and 5(f). On both datasets, sampling 16 or 32 walks with the length 2 or 3 seems sufficient to characterize a temporal node; (3) For efficiency, we limit the number of negatives in Equation 10 instead of calculating $\overline{h}_{v'}$ for all $v' \in \mathcal{G}$ where $v' \neq v$. Specifically, we find that a large number is usually beneficial on both datasets as shown in Figure 5(d), where the performance increases to converge with an increasing number of negatives. In our method, we normally do not have to tune continuous evolution-related hyperparameters, but we provide a detailed study in Appendix D.2.

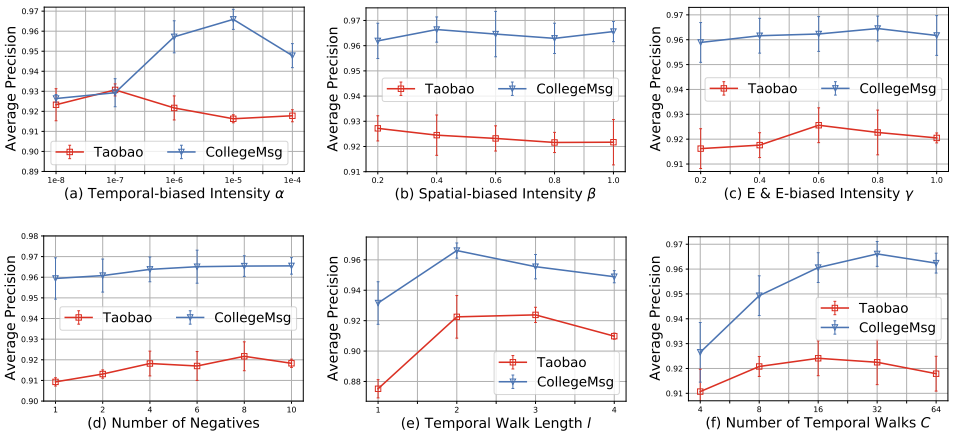


Figure 5: Study on important settings of `NeurTWs`. The performance in predicting *all inductive* interactions is reported.

6 Discussion and Conclusion

Limitations. There are a few limitations for our method. Firstly, a more sophisticated time interval normalization is required. In real-world dynamic graphs, there may be some large time intervals between temporal nodes when constructing temporal walks. Although we propose a simple solution based on logarithmic transformations (see Appendix B.3), there is no theoretical guarantee of stability when solving the continuous evolution process by this normalization trick. Secondly, the calculation of spatial-biased probabilities is computationally heavy. While it is practical to limit the number of spanned temporal neighbors to alleviate the computational burden (see Appendix D.4), we resort to finding a more efficient implementation in the future.

Social Impacts. Continuous-time dynamic graph representation learning benefits a wide range of real-world applications, including but not limited to recommender systems, social network mining, and industrial process modeling. However, there are also some potential negative impacts. For example, `NeurTWs` may learn skewed representations if there are biased patterns in the training data, which may result stereotyped predictions. Also, powerful dynamic graph models may augment harmful activities (e.g., attacking and phishing) on real-world dynamic systems.

Conclusion. We propose a novel and competitive method for modeling CTDGs, which has advantages in three aspects. Firstly, our approach complements existing temporal neighborhood sampling algorithms by simultaneously considering temporal, topological, and tree traversal properties, enabling diverse and expressive dynamic graph motifs to be retrieved. Secondly, the proposed method allows these motifs with irregularly-sampled temporal nodes to be better embedded, where temporal dependencies are now explicitly modeled. Thirdly, our model forms a harder contrastive pretext task to enrich supervision signals. Consequently, the `NeurTWs` method outperforms existing works by large margins, revealing significant application prospects in many real-world scenarios, which will be an exciting focus in our future work.

Acknowledgments and Disclosure of Funding

Some computing resources for this project are supported by MASSIVE [2]. We thank Guangsi Shi for additional computational resources and assistance in model training and deployment.

S. Pan was partially supported by an Australian Research Council (ARC) Future Fellowship (FT210100097). Y.-F. Li was partially supported by the DARPA CCU program (HR001121S0024).

[2] https://www.massive.org.au/

References

[1] Eric Z Chen, Terrence Chen, and Shanhui Sun. Mri image reconstruction via learning optimization using neural odes. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 83–93. Springer, 2020.

[2] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

[3] Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. Neural spatio-temporal point processes. In *International Conference on Learning Representations*, 2020.

[4] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

[5] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Martinez Canedo. Capturing network dynamics using dynamic graph representation learning, March 5 2020. US Patent App. 16/550,771.

[6] Zhen Han, Zifeng Ding, Yunpu Ma, Yujia Gu, and Volker Tresp. Temporal knowledge graph forecasting with neural ode. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, page 8352–8364, 2021.

[7] Zijie Huang, Yizhou Sun, and Wei Wang. Learning continuous system dynamics from irregularly-sampled partial observations. *Advances in Neural Information Processing Systems*, 33:16177–16187, 2020.

[8] Zijie Huang, Yizhou Sun, and Wei Wang. Coupled graph ode for learning interacting system dynamics. In *27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2021*, pages 705–715, 2021.

[9] Boyan Jiang, Yinda Zhang, Xingkui Wei, Xiangyang Xue, and Yanwei Fu. Learning compositional representation for 4d captures with neural ode. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5340–5350, 2021.

[10] Linpu Jiang, Ke-Jia Chen, and Jingqiang Chen. Self-supervised dynamic graph representation learning via temporal subgraph contrast. *arXiv preprint arXiv:2112.08733*, 2021.

[11] Di Jin, Luzhi Wang, Yizhen Zheng, Xiang Li, Fei Jiang, Wei Lin, and Shirui Pan. Cgmn: A contrastive graph matching network for self-supervised graph similarity learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 2101–2107, 2022.

[12] Ming Jin, Yu Zheng, Yuan-Fang Li, Siheng Chen, Bin Yang, and Shirui Pan. Multivariate time series forecasting with dynamic graph neural odes. *arXiv preprint arXiv:2202.08408*, 2022.

[13] Seyed Mehran Kazemi. Dynamic graph neural networks. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 323–349. Springer, 2022.

[14] Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(11):P11005, 2011.

[15] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019.

[16] Günter Last and Mathew Penrose. *Lectures on the Poisson Process*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2017.

[17] Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection; 2014, 2016.

[18] Xu Liu, Yuxuan Liang, Yu Zheng, Bryan Hooi, and Roger Zimmermann. Spatio-temporal graph contrastive learning. *arXiv preprint arXiv:2108.11873*, 2021.

[19] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[20] Shmoolik Mangan and Uri Alon. Structure and function of the feed-forward loop network motif. *Proceedings of the National Academy of Sciences*, 100(21):11980–11985, 2003.

[21] Silvio Micali and Zeyuan Allen Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 200:108–122, 2016.

[22] Michael C Mozer, Denis Kazakov, and Robert V Lindsey. Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110*, 2017.

[23] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 969–976, 2018.

[24] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

[25] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. In *International Workshop on Deep Learning on Graphs*, 2019.

[26] Rui Qian, Tianjian Meng, Boqing Gong, Ming-Hsuan Yang, Huisheng Wang, Serge Belongie, and Yin Cui. Spatiotemporal contrastive video representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6964–6974, 2021.

[27] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML 2020 Workshop on Graph Representation Learning*, 2020.

[28] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in Neural Information Processing Systems*, 32, 2019.

[29] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 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*, pages 519–527, 2020.

[30] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[31] Georg Simmel. *The sociology of georg simmel*, volume 92892. Simon and Schuster, 1950.

[32] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

[33] Sheng Tian, Ruofan Wu, Leilei Shi, Liang Zhu, and Tao Xiong. Self-supervised representation learning on dynamic graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1814–1823, 2021.

[34] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*, 2019.

[35] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

[36] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *International Conference on Learning Representations*, 2021.

[37] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. In *International Conference on Machine Learning*, pages 10432–10441. PMLR, 2020.

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

[39] Yiwen Zhan, Yuchen Chen, Pengfei Ren, Haifeng Sun, Jingyu Wang, Qi Qi, and Jianxin Liao. Spatial temporal enhanced contrastive and pretext learning for skeleton-based action representation. In *Asian Conference on Machine Learning*, pages 534–547. PMLR, 2021.

[40] He Zhang, Bang Wu, Xingliang Yuan, Shirui Pan, Hanghang Tong, and Jian Pei. Trustworthy graph neural networks: Aspects, methods and trends. *arXiv preprint arXiv:2205.07424*, 2022.

[41] Yujia Zhang, Lai-Man Po, Xuyuan Xu, Mengyang Liu, Yexin Wang, Weifeng Ou, Yuzhi Zhao, and Wing-Yin Yu. Contrastive spatio-temporal pretext learning for self-supervised video representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[42] Xin Zheng, Miao Zhang, Chunyang Chen, Chaojie Li, Chuan Zhou, and Shirui Pan. Multi-relational graph neural architecture search with fine-grained message passing. In *IEEE International Conference on Data Mining*, 2022.

[43] Yizhen Zheng, Ming Jin, Shirui Pan, Yuan-Fang Li, Hao Peng, Ming Li, and Zhao Li. Towards graph self-supervised learning with contrastive adjusted zooming. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[44] Yizhen Zheng, Shirui Pan, Vincent Cs Lee, Yu Zheng, and Philip S Yu. Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination. *Advances in Neural Information Processing Systems*, 2022.

[45] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. Tgl: A general framework for temporal gnn training on billion-scale graphs. *Proceedings of the VLDB Endowment*, 15(8):1572–1580, 2022.

[46] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1079–1088, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B.3 for the detailed justification of our batching mechanism discussed in Section 4.5.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix C.3
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix C.3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] We discuss how to generate datasets in Appendix C.1
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Notations

Table 5: Summary of important notations

Notation	Explaination
$\mathcal{G}$	A CTDG composed of a series of temporal interactions, i.e., $\mathcal{G} = \{(e_i, t_i)\}_{i=1}^{N}$
(e, t)	A temporal interaction at time t between nodes u and v where $e := \{u, v\}$
$\mathcal{G}_{u,t}$	A dynamic subgraph composed of the 1-hop temporal neighbors of node u at time t
X_u, X_e	The attributes of node u and interaction e at a particular time
W	A (time-reversed) temporal walk composed of a sequence of temporal nodes
$\widehat{W}$	An anonymous temporal walk where node identities in W are replaced by positional encodings (i.e., relative node identities)
M_u	A set of temporal walks rooted at node u at a particular time
$A(M_u)$	A set of anonymous temporal walks rooted at node u at a particular time
$h_i^{'}$	The latent states of the continuous evolution process at time t_i when encoding $\widehat{W}$
h_i	The activated $h_i^{'}$ with the instantaneous observation $A(w_i) \in \widehat{W}$ at time t_i
$\overline{h}_u$	The learned time-aware representation of node u at a particular time
$A(\cdot)$	The unitary (or binary) temporal walk anonymization
$f(\cdot, \theta)$	The ODE function parameterized by θ in Equation 8
$g(\cdot, \phi)$	The activation function parameterized by ϕ in Equation 9
N	The total number of temporal interactions in $\mathcal{G}$
α, β, γ	The intensities of temporal, spatial, and exploration & exploitation biases
l, C	The walk length and the number of walks rooted at a temporal node

B Additional Model Details

B.1 Tree-Structured Walk Sampling

In practice, we sample walks with a tree-structured sampling algorithm, where temporal, topological, and tree traversal properties are considered simultaneously (Section 4.2). In Algorithm 3, we configure the sampling with a specific hyperparameter ngh that defines the sampling hops and the number of temporal neighbors to be sampled per hop. For example, we sample 32 1-hop and 1 2-hop neighbors if $ngh = [32, 1]$, which forms $C = 32$ walks with the length $l = 2$. In other word, we have $|ngh| = l$ and $\prod_{i=0}^{|ngh|-1} ngh[i] = C$.

Algorithm 3 Tree-Structured Temporal Walk Sampling

Require: Root node w_0, cut time t_0, dynamic graph $\mathcal{G}$, sampling configuration list ngh, temporal-biased intensity α, spatial-biased intensity β, exploration & exploitation-biased intensity γ

Initialize $roots$ = [(w_0, t_0)], L_0 = [(w_0, t_0)], empty lists L_i for i in $1, 2, \cdots, |ngh|$
for i in $1, 2, \cdots, |ngh|$ **do**
  $k = ngh[i]$
  for $root$ in $roots$ **do**
    (w_p, t_p) = $root$, initialize dictionaries $s = \{w : 0 \mid w \in \mathcal{G}_{w_p,t_p}\}$ and $d = \{w : 0 \mid w \in \mathcal{G}_{w_p,t_p}\}$
    $Pr_t = \{w : exp\big(\alpha(t - t_p)\big) \mid (\{w, w_p\}, t) \in \mathcal{G}_{w_p,t_p}\}$ */∗ Temporal-biased probabilities ∗/*
    for $(\{w, w_p\}, t)$ in $\mathcal{G}_{w_p,t_p}$ **do**
      $d[w] = |\mathcal{G}_{w,t}|$
    end for
    $Pr_s = \{w : exp(-\beta/d[w]) \mid w \in \mathcal{G}_{w_p,t_p}\}$ */∗ Spatial-biased probabilities ∗/*
    for j in $1, 2, \cdots, k$ **do**
      $Pr_e = \{w : exp(-\gamma s[w]) \mid w \in \mathcal{G}_{w_p,t_p}\}$ */∗ E&E-biased probabilities ∗/*
      Sample one $(\{w, w_p\}, t) \in \mathcal{G}_{w_p,t_p}$ with prob. $\propto Pr_t, Pr_s, Pr_e$
      $L_i = L_i || (w, t)$
      $s[w]$ += 1
    end for
  end for
  $roots = L_i$
end for
return $\{W_c \mid 1 \leq c \leq C\}$ = Tree2Walk$(L_0, \cdots, L_{|ngh|})$ */∗ Output C walks with length l ∗/*

B.2 Autoregressive Gated Recurrent Unit

Different from the gated recurrent unit (GRU) [4] whose hidden states are simultaneously activated by inputs and previous states, we define the autoregressive GRU as follows:

$$\begin{aligned} z_t &= \sigma(W_z h_{t-1} + b_z), \\ r_t &= \sigma(W_r h_{t-1} + b_r), \\ \widehat{h}_t &= \tanh(W_h(r_t \odot h_{t-1}) + b_h), \\ h_t &= z_t \odot \widehat{h}_t + (1 - z_t) \odot h_{t-1}. \end{aligned} \tag{12}$$

In Equation 12, the hidden states h_t only depend on h_{t-1}, where z_t, r_t, and $\widehat{h}_t$ are update, reset, and candidate activation states. Specifically, W_z, W_r, and W_h are trainable weights. b_z, b_r, and b_h are trainable biases. $\sigma(\cdot)$ and $\tanh(\cdot)$ denote the sigmoid and hyperbolic tangent activation functions.

B.3 Batching and Complexity Analysis

Walk encoding in batches For C different length-l anonymous walks with various time intervals, we normally have to solve $C \times l$ different ODEs to encode each walk, which is computational expensive. Now take C walks with $l = 1$ (i.e., the walks composed of only two temporal nodes) for example, we suppose the timestamps of two (i.e., source and target) nodes are t^c_{start} and t^c_{end} for c in $1, 2, \cdots, C$. Inspired by Chen *et al.* [3], we aim to construct a *substitute variable* s for each continuous evolution process that unify their integral intervals to a specific range, e.g., 0 to 1, so that makes it possible to encode all C walks at once in parallel.

Considering the c-th walk in the above example, we intend to integrate from t^c_{start} to t^c_{end} with the initial states h_0 and an ODE function $f(h_t, t, \theta)$ to learn the latent spatiotemporal dynamics. To integrate from 0 to 1, we transform h_t to $\widetilde{h}_s$ with $s = (t - t^c_{start})/(t^c_{end} - t^c_{start})$. In other word, we have $t = s(t^c_{end} - t^c_{start}) + t^c_{start}$ and the following equation:

$$\widetilde{h}_s = h_{s(t^c_{end} - t^c_{start}) + t^c_{start}} \tag{13}$$

The corresponding ODE function $\widetilde{f}(\widetilde{h}_s, s, \theta)$ then follows:

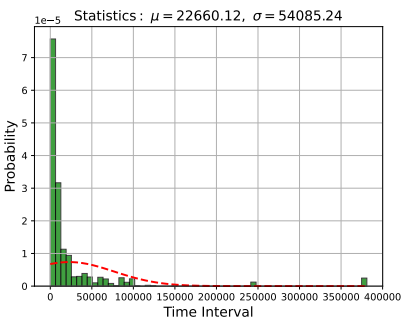
$$\begin{aligned} \widetilde{f}(\widetilde{h}_s, s, \theta) &:= \frac{d\widetilde{h}_s}{ds} = \left.\frac{dh_t}{dt}\right|_{t=s(t^c_{end} - t^c_{start}) + t^c_{start}} \frac{dt}{ds} \\ &= \left. f(h_t, t, \theta)\right|_{t=s(t^c_{end} - t^c_{start}) + t^c_{start}} (t^c_{end} - t^c_{start}) \\ &= f(\widetilde{h}_s, s(t^c_{end} - t^c_{start}) + t^c_{start}, \theta)(t^c_{end} - t^c_{start}) \end{aligned} \tag{14}$$

On this basis, we have the following equation w.r.t. the facts that $\widetilde{h}_0 = h_{t^c_{start}} = h_0$ and $\widetilde{h}_1 = h_{t^c_{end}}$

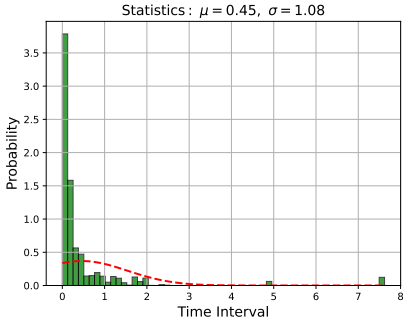
$$h_{t^c_{end}} = \text{ODESolving}(h_0, \widetilde{f}_\theta, 0, 1) = \text{ODESolving}(h_0, f_\theta, t^c_{start}, t^c_{end}), \tag{15}$$

where ODESolving$(\cdot)$ denotes a black-box ODE solver, e.g., Euler or Runge-Kutta 3/8 method. With the above reparameterization trick, we can solve C different continuous evolution processes in parallel, thus allowing our method to be trained more efficiently.

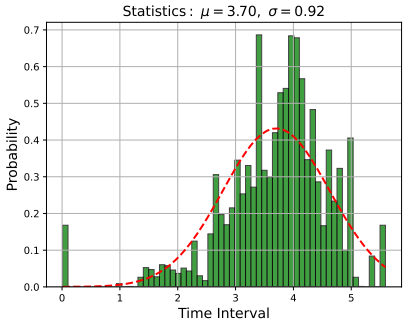
Time interval normalization There may be some large time intervals between two temporal nodes in real-world dynamic graphs, e.g., Figure 6(a), which makes solving the continuous evolution process (i.e., Equation 8) intractable. A good idea to address this challenge is to scale time intervals without violating the inductive biases that the relative differences between time intervals reflect essential temporal clues. On this basis, various transformations can be applied, such as logarithmic transformations or divided by a large constant, to scale down the range of time intervals and make solving Equation 8 tractable. In practice, we empirically find that using the base 10 logarithmic scaling, as shown in Figure 6(c), is sufficient for our method to achieve good performances on dynamic graph modeling. Compared with other alternatives, e.g., divided by a large constant as shown in Figure 6(b), the logarithmic transformation with a properly selected base can scale down the



(a) Distribution of raw time intervals in seconds.



(b) Distribution after constant-based scaling with the denominator 5×10^4.



(c) Distribution after logarithmic scaling with the base 10.

Figure 6: Average time intervals between temporal nodes in a batch of walks on the Reddit dataset. Specifically, we let batch size $B = 32$ and the number of walks $C = 64$ per root node with the walk length $l = 2$.

range of time intervals while well preserving the variances between most small intervals. Although the variances between very large intervals are compressed in such a case, we conjecture the key is retaining the relative differences between small and large time intervals, which reveals critical temporal patterns in dynamic graph modeling.

Complexity analysis We analyse the time complexity of the main components in `NeurTWs`, as shown in Table 6.

Table 6: Time complexity analysis

Component	**Time Complexity**
Temporal-biased walk sampling	$\mathcal{O}(BC)$
Spatial-biased walk sampling	$\mathcal{O}(BCk_s)$
E&E-biased walk sampling	$\mathcal{O}(BC)$
Subtotal (Algorithm 3 and anonymization)	$\mathcal{O}(BC(k_s + l))$
Continuous evolution	$\mathcal{O}(lFd_h^2)$
Instantaneous activation	$\mathcal{O}(ld_h^2)$
Subtotal (Algorithm 2)	$\mathcal{O}(lFd_h^2)$
Total (Algorithm 4)	$\mathcal{O}\big(K(BC(k_s + l) + lFd_h^2)\big)$

We start from analysing the time complexity of the proposed temporal walk sampling based on Algorithm 3 with the following assumptions: (1) We have a batch of B interactions; (2) The sampling configuration $ngh = [k_1, k_2, \cdots, k_l]$ s.t. the number of walks $C = \prod_{i=1}^{|ngh|} k_i$ and the walk length $l = |ngh|$; (3) The time of neighborhood finding is constant; (4) The average number of spanned

temporal neighbors is k_s. On this basis, the time complexity of temporal-biased, spatial-biased, and exploration & exploitation-biased walk sampling are $\mathcal{O}(BC)$, $\mathcal{O}(BCk_s)$, and $\mathcal{O}(BC)$, respectively. Thus, the time complexity of our sampling algorithm is $\mathcal{O}(BCk_s)$. For walk anonymization, we analyze the upper bound time complexity based on a simple brute-force implementation. We first identify unique nodes among C temporal walks with the length l by traversing all nodes in walks. Then, we traverse again to acquire exact node positional encodings, thus resulting in the complexity of $\mathcal{O}(Cl)$ for processing an interaction in a batch.

We then analyse the time complexity of the proposed continuous evolution and instantaneous activation processes with the aforementioned and the following additional assumptions: (1) We instantiate $g(\cdot, \phi)$ and $f(\cdot, \theta)$ with a standard and autoregressive GRU (See Appendix B.2), respectively.; (2) We assume the input and hidden dimensions are d_i and d_h, i.e., $A(w_i) \in \mathbb{R}^{d_i \times 1}$ and $A^{'}(w_i), h_i^{'}, h_i \in \mathbb{R}^{d_h \times 1}$; (3) We assume the number of function evaluations (i.e., how many times the ODE function is called in a single forward pass when solving an ODE) in the continuous evolution process is F. On this premise, the time complexity of solving the continuous evolution process over $B \times C$ walks is $\mathcal{O}(BClFd_h^2)$. However, the aforementioned batching mechanism allows those different ODEs to be solved in parallel, which reduces the time complexity to $\mathcal{O}(lFd_h^2)$. Regarding the instantaneous activation process, the time complexity is $\mathcal{O}(ld_h^2)$ when training in batches.

Since our model is trained by optimizing a contrastive objective function, the overall time complexity of our method is $\mathcal{O}\big(K(BC(k_s + l) + lFd_h^2)\big)$, where K denotes the number of negatives.

B.4 Model Training

Algorithm 4 Training `NeurTWs`

Require: Training subgraph $\mathcal{G}_{trn}$, batch size B, sampling configuration ngh, temporal intensity α, spatial intensity β, exploration & exploitation intensity γ, number of negatives K, learning rate η

Initialize $epoch = 1$
repeat
  Split all interactions in $\mathcal{G}_{trn}$ into m batches $\widetilde{\mathcal{E}} = \{\widetilde{\mathcal{E}}_i \mid 1 \leq i \leq m,\ |\widetilde{\mathcal{E}}_i| = B\}$
  for i in $1, 2, \cdots, m$ **do**
    Initialize $\overline{h}_{V'}^{K} = \{\}$
    / TWS$(\cdot)$ stands for temporal walk sampling (see Algorithm 3) */*
    Temporal walk sampling $M_U = \{M_{u_j} \leftarrow \text{TWS}(u_j, t_j, \mathcal{G}_{trn}, ngh, \alpha, \beta, \gamma) \mid (\{u_j, v_j\}, t_j) \in \widetilde{\mathcal{E}}_i\}$, $M_V = \{M_{v_j} \leftarrow \text{TWS}(v_j, t_j, \mathcal{G}_{trn}, ngh, \alpha, \beta, \gamma) \mid (\{u_j, v_j\}, t_j) \in \widetilde{\mathcal{E}}_i\}$
    Anonymization $A(M_U) = \{A(M_{u_j}) \mid M_{u_j} \in M_U\}$, $A(M_V) = \{A(M_{v_j}) \mid M_{v_j} \in M_V\}$
    / TWE$(\cdot)$ stands for walk encoding (see Algorithm 2). Pool$(\cdot)$ denotes sum-pooling */*
    Walk encoding and pooling $\overline{h}_U = \{\overline{h}_{u_j} \leftarrow \text{Pool}\Big(\text{TWE}\big(A(M_{u_j})\big)\Big) \mid A(M_{u_j}) \in A(M_U)\}$, $\overline{h}_V = \{\overline{h}_{v_j} \leftarrow \text{Pool}\Big(\text{TWE}\big(A(M_{v_j})\big)\Big) \mid A(M_{v_j}) \in A(M_V)\}$
    / Negative sampling and calculate the time-aware node embedding of negatives */*
    for k in $1, 2, \cdots, K$ **do**
      Randomly sample a batch of nodes $V' = \{v_j' \in \mathcal{G}_{trn} \mid 1 \leq j \leq B,\ (\{u_j, v_j'\}, t_j) \notin \mathcal{G}_{trn}\}$
      Temporal walk sampling $M_{V'} = \{M_{v_j'} \leftarrow \text{TWS}(v_j', t_j, \mathcal{G}_{trn}, ngh, \alpha, \beta, \gamma) \mid v_j' \in V'\}$
      Anonymization $A(M_{V'}) = \{A(M_{v_j'}) \mid M_{v_j'} \in M_{V'}\}$
      Walk encoding and pooling $\overline{h}_{V'} = \{\overline{h}_{v_j'} \leftarrow \text{Pool}\Big(\text{TWE}\big(A(M_{v_j'})\big)\Big) \mid A(M_{v_j'}) \in A(M_{V'})\}$
      $\overline{h}_{V'}^{K} = \overline{h}_{V'}^{K} \,||\, \overline{h}_{V'}$
    end for
    / Calculate the contrastive loss w.r.t. Equation 10 */*
    Compute $\mathcal{L} = loss(\overline{h}_U, \overline{h}_V, \overline{h}_{V'}^{K})$ and the stochastic gradients of model parameters w.r.t. $\mathcal{L}$
    Update model parameters w.r.t. their gradients and the learning rate η
  end for
  $epoch \mathrel{+}= 1$
until convergence

C Experimental Setting

C.1 Datasets

We list the introduction of six datasets used in this paper as follows. We also discuss how they are preprocessed for use in our method.

- **CollegeMsg** [3] collects the message interactions between the users in an online social media platform at the University of California, Irvine. Specifically, this dataset consists of 59,835 message interactions and 1,899 unique users, where these unattributed temporal interactions are in the format of (u, v, t), denoting that user u sent a message to user v at time t. Since the format of the interactions in this dataset naturally matches the definition of continuous-time dynamic graphs in Section 3, so we do not need to preprocess the dataset intentionally. We provide the ready-to-use dataset files with our code, as well as a script to transform the raw dataset file to the data files used in our model.

- **Enron** [4] is an email network dataset consisting of around 0.5 million email interactions between about 150 senior employees in a corporation over several years. In this paper, we build the Enron dataset based on the preprocessed version in [36] by using the first half of interactions, which composes of 62,617 unattributed email interactions and 143 unique employees. We provide the preprocessed dataset files with our code.

- **Taobao** [5] is an user behavior dataset that originally collected for the research of recommendation systems, which consists of 100,150,807 user-item interactions, 987,994 users, and 4,162,024 items. Notably, each interaction in this dataset has a specific action type, i.e., a user browses/buys/favorites an item or adds it to the shopping cart. To incorporate this information, we one-hot encode each interaction's action type as the interaction attributes. In this paper, we build the Taobao dataset with a subset of the original user-item interaction network, which composes of 77,436 interactions and 64,703 unique nodes. We provide the preprocessed dataset files with our code, as well as a script to preprocess and transform the raw dataset files to the data files used in our model.

- **MOOC** [6] is an user action dataset that has a collection of actions taken by the users on the MOOC platform. This dataset consists of 7,047 users, 97 items, and 411,749 attributed interactions over the duration of about a month. Specifically, the user-item interactions in this dataset denote the access behavior of students to online course units. Similar to CollegeMsg, the format of this dataset naturally matches our definition of continuous-time dynamic graphs, so we have not preprocessed it intentionally. We provide the ready-to-use dataset files with our code.

- **Wikipedia** [7] is an user action dataset that has a set of interactions between users and wikipedia pages. This dataset records 157,474 attributed interactions in one month between 8,227 users and 1,000 pages. Specifically, an interaction denotes that an user edits a page, where the editing texts have converted into LIWC-feature vectors. Similar to CollegeMsg and MOOC, we have not preprocessed this dataset intentionally. Also, we provide the ready-to-use dataset files with the code.

- **Reddit** [8] is an user action dataset that collects one month of posts made by different users on the Reddit platform. Similar to Wikipedia, an interaction in this dataset denotes that an user makes a post on subreddits, where the texts are embedded into LIWC-feature vectors. Specifically, this dataset consists of 672,447 interactions, 10,000 most active users and 984 subreddits. We have not preprocessed this dataset intentionally, and we also provide the dataset files with the code.

[3] https://snap.stanford.edu/data/CollegeMsg.html
[4] http://www.cs.cmu.edu/~enron/
[5] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649&userId=1
[6] http://snap.stanford.edu/jodie/mooc.csv
[7] http://snap.stanford.edu/jodie/reddit.csv
[8] http://snap.stanford.edu/jodie/wikipedia.csv

C.2 Baselines

We list the introduction of baseline methods as follows.

- **CTDNE**[23] extends the static network embedding to dynamic graphs, where temporal random walks have been proposed with the skip-gram model to learn node representations.
- **JODIE**[15] updates latent states of two nodes in an interaction with two mutually-influenced recurrent neural networks, where the future node embedding trajectories can be estimated.
- **DyRep**[34] integrates sequence models with an attentive message passer, where 2-hop temporal neighborhood information is leveraged to learn time-aware node embeddings.
- **TGAT**[38] samples and attentively aggregates the information from k-hop temporal neighbors to learn time-aware node embeddings. In this method, the time information is preserved by the proposed random Fourier time encodings when conducting the message passing.
- **TGN**[27] proposes a general message passing-based framework to learn on CTDGs with a node memory updating mechanism, which inherits and combines the key designs in JODIE and TGAT.
- **CAWs**[36] learn interaction representations by encoding and aggregating anonymous temporal walks with a recurrent neural network and a pooling module.

Specifically, we detail how baseline models are tuned in link prediction tasks as follows.

- We adapt TGAT [9] to our unified evaluation framework, and mainly tune the following important hyperparameters: (1) For the degree of neighborhood sampling, we search the optimal values in $\{8, 16, 32, 64\}$; (2) We tune the model layers in $\{1, 2, 3\}$; (3) We use the default product attention and tune the number of attention heads in $\{2, 4, 6\}$; (4) For the dimensions of node and time embeddings, we search the optimal configuration in $\{16, 32, 64, 100\}$. Regarding the rest of hyperparameters, we use the default setting in the code of TGAT.
- For JODIE, DyRep, and TGN, we adapt the implementation [10] to our unified evaluation framework. Specifically, we mainly follow the default setting when executing JODIE on our datasets, where the dimension of node embeddings is searched in $\{32, 64, 128, 256\}$. For DyRep, we tune the degree of neighborhood sampling in $\{10, 16, 32, 64\}$ and the number of graph attention layers in $\{1, 2, 3\}$. We also search the optimal values of these two hyperparameters in TGN, where additional hyperparameters are tuned as well: (1) We tune the number of attention heads in $\{2, 4, 8\}$; (2) We tune the dimension of node, time, and message embeddings in $\{16, 32, 64, 100\}$; (3) We tune the dimension of node memories in $\{4, 16, 32, 64, 172\}$.
- Our method shares the same evaluation pipeline with CAWs [11], for which we mainly tune the following hyperparameters based on their default setting: (1) We tune the walk length in $\{1, 2, 3\}$ and the number of walks in $\{16, 32, 64, 128\}$; (2) We tune the time decay in $\{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$; (3) We tune the model with the unitary and binary anonymization, where the best performances are reported. We follow their default setting for the rest of the hyperparameters except for the walk pooling, which is set to summation as same as in our method for simplicity and fair comparison.

C.3 Implementation Details

Our code is available at https://github.com/KimMeen/Neural-Temporal-Walks, where we provide detailed instructions for dataset preparation and model training.

We first introduce the general setting of `NeurTWs` when training on all three datasets. Specifically, our models are trained with the learning rate 10^{-4} and batch size 32. We set the maximum number of epochs to 50, but the model training usually converges and is early stopped within the first 20 epochs.

For important hyperparameters controlling the walk sampling and contrasitve learning, we summarize their grid search ranges in Tables 7 and 8.

[9] https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-graphs
[10] https://github.com/twitter-research/tgn
[11] https://github.com/snap-stanford/CAW

Table 7: Search ranges of important hyperparameters on two unattributed datasets

Hyperparameter	CollegeMsg	Enron
Walk length l	$\{1, 2, 3\}$	$\{1, 2, 3, 4\}$
Number of walks C	$\{16, 32, 64\}$	$\{16, 32, 64\}$
Number of negatives	$\{1, 3, 6, 9, 18\}$	$\{2, 4, 6, 8, 16\}$
Temporal bias intensity α	$\{10^{-6}, 10^{-5}, 10^{-4}\}$	$\{10^{-7}, 10^{-6}, 10^{-5}\}$
Spatial bias intensity β	$\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$	$\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$
E&E trade-off intensity γ	$\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$	$\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$

Table 8: Search ranges of important hyperparameters on two attributed datasets

Hyperparameter	Taobao	MOOC
Walk length l	$\{1, 2, 3\}$	$\{1, 2, 3\}$
Number of walks C	$\{16, 32, 64\}$	$\{16, 32, 64\}$
Number of negatives	$\{2, 4, 6, 8, 16\}$	$\{2, 4, 6, 8\}$
Temporal bias intensity α	$\{10^{-7}, 10^{-6}, 10^{-5}\}$	$\{10^{-6}, 10^{-4}\}$
Spatial bias intensity β	$\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$	$\{10^{-1}, 5 \times 10^{-1}, 1\}$
E&E trade-off intensity γ	$\{0, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$	$\{5 \times 10^{-1}, 1\}$

For the rest of hyperparameters, we set them as follows: (1) We use the fixed-step ODE solver (i.e., Runge-Kutta 3/8 method) with the step size $= 0.125$ by default without specific tuning. Adaptive solvers (e.g., Dormand–Prince method) can also be used in our method; (2) The hidden dimensions of the position encoding has not tuned and set to 108 by default; (3) For the walk pooling, we use the summation by default for simplicity. More details can be found in our code, where we provide the scripts with the default settings to run experiments on three datasets.

All experiments are conducted on a Linux cluster with four computing nodes (for each we have $4\times$ RTX6000 24GB, $1\times$ Intel Xeon Silver 4214R, and 175 GB system memory) and a Linux server ($4\times$ GeForce RTX 2080 Ti 11GB, $1\times$ Intel Core i9-9900X, and 94GB system memory).

D Additional Experimental Results

D.1 Performances in Average Precision

See Table 9 for detailed transductive and inductive link prediction results w.r.t. AP.

D.2 Study on Continuous Evolution Process

We study the effectiveness of the proposed continuous evolution process in modeling temporal dependencies when encoding anonymous temporal walks, and further compare it with other two widely adopted strategies, i.e., exponential decay [22] and random Fourier time encoding [38]. We summarize the results in Table 10. Specifically, we use GRU as the RNN backbone without considering the time information associated with each temporal node in $\widehat{W}$ (i.e., Standard RNN). On this basis, we equip GRU with the exponential decay mechanism (i.e., RNN with exponential decay), which is defined as follows based on Equation 9 in the main body of the paper:

$$h_i = \text{GRUCell}(h_{i-1} \cdot exp(-\tau \Delta_t), A'(w_i), \phi), \tag{16}$$

where $\Delta_t = t_i - t_{i-1}$ and τ is a hyperparameter to control the rate of decay. In our experiments, we tune τ in $\{10^{-9}, 10^{-8}, 10^{-7}\}$. Apart from this, time encoding is another strategy to incorporate the time information, which is defined below [38]:

$$\begin{aligned} h_i &= \text{GRUCell}(h_{i-1}, A'(w_i) \,||\, \langle \Phi_d(t_{i-1}), \Phi_d(t_i) \rangle, \phi), \\ \Phi_d(t) &= \sqrt{\frac{1}{d}} \big[\cos(\omega_1 t) \sin(\omega_1 t), \cdots, \cos(\omega_d t) \sin(\omega_d t) \big], \end{aligned} \tag{17}$$

where $\Phi_d(\cdot)$ denotes the time encoding with the dimensions of d. $\langle \cdot \rangle$ and $||$ are inner product and concatenation operation. We implement the time encoding-based variant (i.e., RNN with time encoding) based on Equation 17.

Table 9: Transductive and inductive link prediction performances w.r.t. AP. We use **bold font** and underline to highlight the best and second best performances. `NeurTWs†` is a vairant of our method with the binary anonymization.

Task		Methods	CollegeMsg	Enron	Taobao	MOOC
Transductive		JODIE [15]	0.5329 ± 0.029	0.8467 ± 0.023	0.8412 ± 0.013	0.6386 ± 0.021
		DyRep [34]	0.5029 ± 0.023	0.8237 ± 0.022	0.8397 ± 0.022	0.5723 ± 0.005
		TGAT [38]	0.7212 ± 0.005	0.6893 ± 0.008	0.5598 ± 0.009	0.6704 ± 0.032
		TGN [27]	0.9007 ± 0.003	0.8876 ± 0.016	0.8677 ± 0.017	0.7347 ± 0.036
		CAWs [36]	0.9255 ± 0.001	0.9437 ± 0.002	0.8829 ± 0.001	0.7011 ± 0.062
		`NeurTWs`	0.9612 ± 0.001	0.9503 ± 0.008	**0.9030 ± 0.009**	**0.7593 ± 0.027**
		`NeurTWs†`	**0.9773 ± 0.003**	**0.9664 ± 0.010**	0.8817 ± 0.012	0.7228 ± 0.033
Inductive	New-Old	JODIE [15]	0.4413 ± 0.052	0.8190 ± 0.024	0.8202 ± 0.005	0.6914 ± 0.010
		DyRep [34]	0.4088 ± 0.034	0.7322 ± 0.018	0.8237 ± 0.008	0.5633 ± 0.014
		TGAT [38]	0.6965 ± 0.006	0.6324 ± 0.033	0.5738 ± 0.021	0.6408 ± 0.014
		TGN [27]	0.8632 ± 0.002	0.7046 ± 0.128	0.8792 ± 0.013	0.7271 ± 0.002
		CAWs [36]	0.9179 ± 0.014	**0.9577 ± 0.005**	0.8960 ± 0.007	0.7602 ± 0.022
		`NeurTWs`	0.9653 ± 0.010	0.9466 ± 0.003	**0.9294 ± 0.012**	**0.7730 ± 0.006**
		`NeurTWs†`	**0.9721 ± 0.008**	0.9488 ± 0.009	0.9063 ± 0.011	0.7617 ± 0.008
	New-New	JODIE [15]	0.4427 ± 0.091	0.7332 ± 0.028	0.8452 ± 0.014	**0.8263 ± 0.015**
		DyRep [34]	0.5502 ± 0.102	0.5966 ± 0.091	0.8423 ± 0.019	0.4484 ± 0.047
		TGAT [38]	0.6983 ± 0.036	0.6593 ± 0.030	0.5660 ± 0.024	0.6324 ± 0.012
		TGN [27]	0.7568 ± 0.112	0.9418 ± 0.016	0.8949 ± 0.019	0.6547 ± 0.065
		CAWs [36]	0.9243 ± 0.010	0.9697 ± 0.003	0.9005 ± 0.005	0.7688 ± 0.037
		`NeurTWs`	0.9702 ± 0.005	**0.9853 ± 0.013**	**0.9252 ± 0.002**	0.8251 ± 0.017
		`NeurTWs†`	**0.9780 ± 0.008**	0.9845 ± 0.016	0.9133 ± 0.014	0.8067 ± 0.013

Table 10: Study on different strategies to model temporal dependencies in temporal walk encoding (Section 4.3). The performance in predicting *all inductive* interactions is reported.

Configuration	CollegeMsg		Taobao	
	AUC	AP	AUC	AP
Standard RNN	0.868 ± 0.02	0.898 ± 0.01	0.860 ± 0.04	0.901 ± 0.02
RNN with exponential decay	0.915 ± 0.03	0.925 ± 0.03	0.923 ± 0.01	0.920 ± 0.01
RNN with time encoding	0.910 ± 0.02	0.903 ± 0.01	0.889 ± 0.01	0.906 ± 0.02
Continuous evolution	**0.958 ± 0.01**	**0.966 ± 0.01**	**0.938 ± 0.02**	**0.933 ± 0.02**

From the results in Table 10, we have the following observations: (1) All variants and our method improve the performance of standard RNN, demonstrating the importance of leveraging time information when learning on CTDGs; (2) The proposed continuous evolution process significantly surpasses the exponential decay and time encoding approaches, proving its effectiveness in capturing the temporal dependencies; (3) Although time encoding helps the model be aware of critical time information, its effectiveness is limited compared with our method.

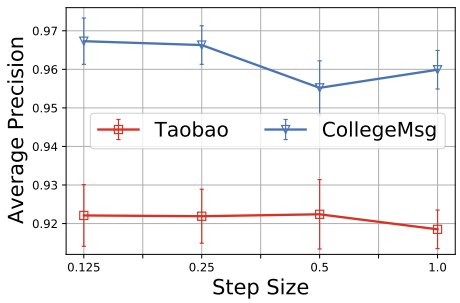
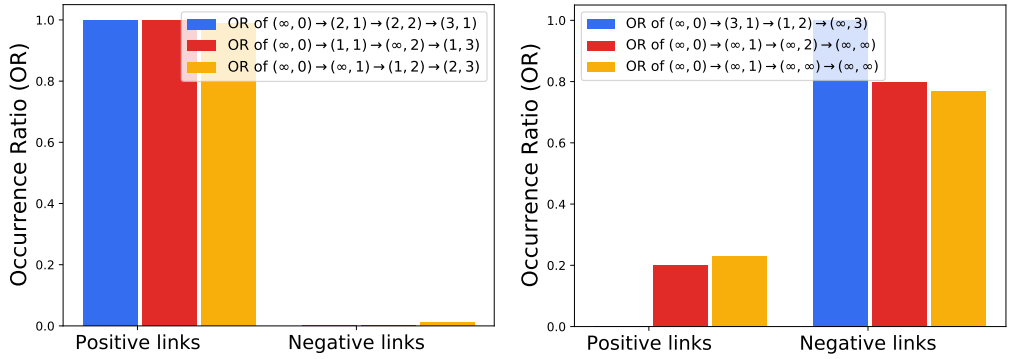


Figure 7: The performance w.r.t. different step sizes in solving the continuous evolution process. The average precision in predicting *all inductive* interactions is reported.

Although the continuous evolution process is less tuned in our method as mentioned in Appendix C.3, we study how the NeurTWs method is affected w.r.t. the number of function evaluations (i.e., the number of times the ODE function is called in a single forward pass when solving an ODE). The experimental results are in Figure 7, where we use the Euler method with different step sizes for illustration purposes. Notably, the Euler method is a fixed-step solver, and all ODEs to be solved are restricted to a specific integral interval (i.e., 0 to 1, see Appendix B.3). Thus, the number of function evaluations depends only on the step size and is inversely proportional to it. From the results in Figure 7, we find that the model performance decreases slightly as the step size increases when solving ODEs. This phenomenon makes sense, as a smaller step size equates to more function evaluations, which allow for more precise learning of the underlying spatiotemporal dynamics in dynamic graph motifs, but at the cost of higher complexity. In other words, our method allows a trade-off between model accuracy and efficiency.

D.3 Visualization



(a) The occurrence ratio of top-3 highest-scored motifs in positive and negative samples.

(b) The occurrence ratio of top-3 lowest-scored motifs in positive and negative samples.

Figure 8: Visualizing the most discriminative dynamic graph motifs learned by NeurTWs† on the CollegeMsg dataset, where their occurrence ratios in positive and negative samples are compared.

We follow the method described in [36] to visualize the most discriminative dynamic graph motifs learned by our method with the binary anonymization (i.e., NeurTWs†) on the CollegeMsg dataset.

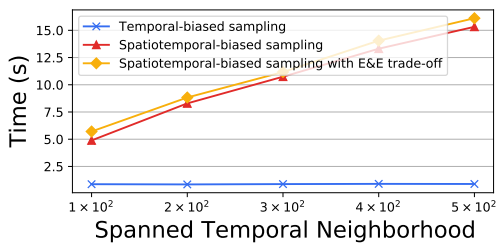
We first illustrate how a motif can be represented in a sequence of coordinates, which essentially characterizes its shape in a coordinate system w.r.t. the extracted temporal walks rooted at two nodes of an interaction. Specifically, given a sampled interaction $(\{u, v\}, t)$ from the dynamic graph $\mathcal{G}$, we sample two sets of temporal walks M_u and M_v rooted at node u and v, respectively. Recall the binary anonymization defined in Section 4.2, $A(w; M_u)[i]$ and $A(w; M_v)[i]$ count the number of walks in M_u and M_v that have node w appearing in position i, where $i \in \{0, \cdots, l\}$. On this basis, we have the shortest path between node w and a root node u (w.r.t. the subgraph composed of the walks in M_u) defined as $d_{wu} := \min\{i | A(w; M_u)[i] > 0\}$. We can define d_{wv} in a similar way with $A(w; M_v)$. Thus, the coordinate mapping $A(w; M_u, M_v) \to (d_{wu}, d_{wv})$ can be viewed as a way to map the position encoding of node w to its coordinate in a coordinate system defined over M_u and M_v. As a result, we can obtain the shapes of anonymous temporal walks by mapping each node's position encoding to its relative coordinate. Specifically, ∞ in a coordinate, e.g., $(\infty, 0)$, denotes that the target node of an interaction does not appear in the temporal walks rooted at the source node of this interaction. A special case (∞, ∞) denotes a nonexistent node, which is used in walk padding to make sure all walks have the same length l when there are no temporal neighbors can be sampled.

We then describe how most discriminative motifs can be identified. Given a sampled interaction $(\{u, v\}, t)$ from the dynamic graph $\mathcal{G}$ and a well-trained model of NeurTWs† with a linear downstream link predictor $\Psi^\intercal(\overline{h}_u \,||\, \overline{h}_v) := \Psi^\intercal \sum_{c=1}^{2C} \text{enc}(\widehat{W}_c)$ for $\widehat{W}_c \in A(M_u) \cup A(M_v)$, we can obtain the predictive score of each motif $\widehat{W}_c$ that contributes to the final prediction of this interaction, i.e., $\Psi^\intercal \text{enc}(\widehat{W}_c)$. In our experiments, we visualize the dynamic graph motifs with the top-3 highest and

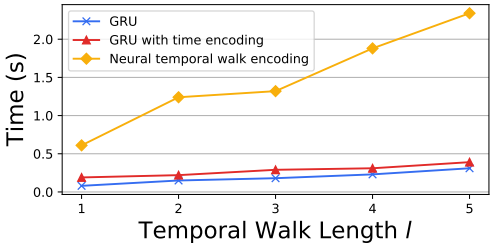
lowest predictive scores in the testing set of the CollegeMsg dataset, as well as their normalized occurrence ratios in all positive and negative samples during the model inference.

From the results in Figure 8, we have the following observations: (1) A general dynamic law can be observed among the shapes of highest-scored motifs; that is, two temporal nodes surrounded by some common dynamic graph motifs are more likely to interact. For example, the shape of the highest-scored motif $(\infty, 0) \rightarrow (2, 1) \rightarrow (2, 2) \rightarrow (3, 1)$ indicates that two temporal nodes share some common 3-hop temporal neighbors because there is no ∞ in the second to fourth positions of the shape of this motif. On the contrary, two temporal nodes with few overlapped dynamic graph motifs are unlikely to interact, e.g., $(\infty, 0) \rightarrow (\infty, 1) \rightarrow (\infty, 2) \rightarrow (\infty, \infty)$. Thus, our method allows informative dynamic graph motifs and the underlying dynamic laws to be extracted and captured effectively; (2) These three highest-scored motifs commonly appear in positive samples where interactions exist between temporal node pairs but rarely appear in negative samples (see Figure 8(a)). In contrast, for those lowest-scored motifs, they appear more frequently in negative samples (see Figure 8(b)). Therefore, the discriminative power of our method is proven to be strong, which captures essential dynamic laws to characterize temporal nodes where downstream tasks can be effectively conducted.

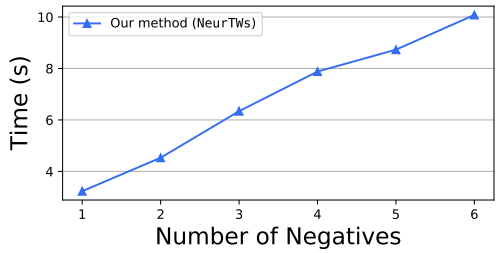
D.4 Complexity Evaluation



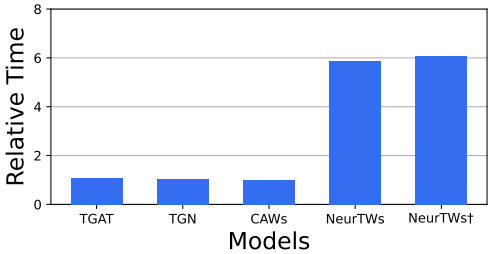
(a) Average walk sampling runtime in a batch w.r.t. the number of spanned temporal neighborhood.



(b) Average walk encoding runtime in a batch w.r.t. the temporal walk length.



(c) Average runtime of our method in a batch w.r.t. the number of negatives.



(d) Related average runtime of different methods per batch based on their optimized settings.

Figure 9: Study the efficiency of our method from different perspectives on the Enron dataset.

We empirically study the efficiency of our method by recording and comparing the runtime of main components in temporal walk sampling and encoding. We also plot the average runtime of `NeurTWs` with different number of negatives during the optimization and the relative runtime comparison between it and other baselines under their optimized settings. The experimental results are in Figure 9, which is consistent with our complexity analysis in Appendix B.3.

We first compare the runtime of the proposed spatiotemporal-biased walk sampling with its two ablation variants on the Enron dataset under the different number of spanned temporal neighbors per node, as shown in Figure 9(a). Specifically, we use the default model setting on Enron that samples 64 walks with the length 2, but the number of negatives is set to 1 for a more straightforward comparison. The number of spanned temporal neighbors is defined as $|\mathcal{G}_{w_p,t_p}|$ in Algorithm 3. From the results in Figure 9(a), we have three observations: (1) Temporal-biased walk sampling has a constant runtime w.r.t. the number of spanned temporal neighbors. This observation can be confirmed by our complexity analysis summarized in Table 6, where the temporal-biased sampling is only in

proportion to the batch size and the number of walks to be sampled; (2) The spatiotemporal-biased walk sampling is in proportion to the number of spanned temporal neighbors, which proves our analysis in Appendix B.3 that its time complexity is $\mathcal{O}(BCk_s)$, where k_s is the x-axis of Figure 9(a); (3) Our spatiotemporal-biased walk sampling with the exploration & exploitation trade-off has a similar time complexity compared to its variant without considering this trade-off. In general, although the calculation of the proposed spatial-biased probabilities induces higher time complexity when sampling temporal walks, it helps extract diverse and expressive dynamic graph motifs, e.g., disabling this mechanism hurts the performance drastically on the Taobao dataset (e.g., ablation 2 in Table 4). To alleviate the computational burden when modeling dynamic graphs with a large average number of spanned temporal neighbors, we find it is practical to limit k_s by selecting a set of (potentially informative) candidates to explore, e.g., we may choose these candidate neighbors randomly or based on prior knowledge. A concrete example is Enron with a higher k_s than other datasets; we empirically find that limiting the number of first- and second-order neighbors with small upper bound (e.g., 64×5 and 1×8) in a temporal-biased manner does not affect the performances of our method significantly.

We then compare the runtime of the proposed neural temporal walk encoding with one of its ablation variants (i.e., GRU, which essentially forms the instantaneous activation process without the continuous evolution process) and an in-demand design (i.e., GRU with time encoding, which adopts random Fourier time encodings as a part of input features of GRU to preserve the time information when learning walk embeddings). The experimental results are in Figure 9(b), where we use the setting as mentioned above except for the walk length, which is the x-axis of this plot. Specifically, we find that the runtime of all methods is in proportion to the walk length, which matches the complexity analysis in Appendix B.3. Compared to GRU, the proposed neural temporal walk encoding has a higher time complexity which is also in proportion to the number of function evaluations F in the continuous evolution process (see Table 6). In Figure 9(b), we find that the runtime of our method is about 7.8 times that of the GRU, which is close to our training setting where $F = 8$. Although the proposed continuous evolution process results in a higher runtime of our method in this example compared with the time encoding approach, it demonstrates significantly better capabilities in digesting the time information (see Table 10). Besides, the proposed continuous evaluation process supports the trade-off between model accuracy and efficiency, allowing our encoding module to have the same time complexity as the time encoding-based GRU (i.e., $\mathcal{O}(ld_h^2)$ by letting the step size $= 1$) without compromising too much modeling precision (see Figure 7).

In Figure 9(c), we display the runtime of the entire model in a batch w.r.t. the number of negatives, which proves our justification in Appendix B.3 that the time complexity of our method is in proportion to the number of negatives during the optimization under a certain model configuration (see Table 6).

In Figure 9(d), we compare the relative runtime between our method and other baselines under their optimized settings on Enron. Although the core components in `NeurTWs` induce higher time complexity, they significantly improve existing approaches from various aspects in terms of the modeling precision. It is worth noting that our method explicitly supports trading precision for speed to alleviate the computational burden. For example, limiting the number of spanned temporal neighbors is practical to reduce runtime (see Figure 9(a)) without losing too much modeling precision, as discussed above. Also, reducing the number of function evaluation F (e.g., increasing the step size when solving Equation 8 with a fixed-step solver) brings significant speedups, which reduces the complexity of neural temporal walk encoding (see Table 6). For the optimization, limiting the number of negatives reduces the model runtime as well.