5-2023

Graph neural point process for temporal interaction prediction

Wenwen XIA

Yuchen LI
Singapore Management University, yuchenli@smu.edu.sg

Shengdong LI

Graph Neural Point Process for Temporal Interaction Prediction

Wenwen Xia, Yuchen Li, and Shenghong Li, *Senior Member, IEEE*

Abstract—Temporal graphs are ubiquitous data structures in many scenarios, including social networks, user-item interaction networks, etc. In this paper, we focus on predicting the exact time of future interactions between node pairs on a temporal graph. This problem can support interesting applications including time-sensitive items recommendation, congestion prediction on road networks, etc. We present the Graph Neural Point Process (GNPP) to tackle this problem. GNPP relies on the graph neural message passing and the temporal point process framework. Most previous graph neural models devised for temporal graphs either utilize the chronological order information or rely on specific point process models, ignoring the exact timestamps and complicated temporal patterns. In GNPP, we adapt a time encoding scheme to map real-valued timestamps to a high-dimensional vector space so that the temporal information can be modeled precisely. Further, GNPP considers the structural information of graphs by conducting message passing aggregation on the constructed line graph. The obtained representation defines a neural conditional intensity function that models events' generation mechanisms for predicting interactions' time between node pairs. We evaluate this model on several synthetic and real-world temporal graphs where it outperforms recently proposed neural point process models and graph neural models devised for temporal graphs. We further conduct ablation comparisons and visual analyses to shed some light on the learned model and understand the functionality of important components comprehensively.

Index Terms—Temporal graphs, Graph neural networks, Point processes

1 Introduction

Graphs are ubiquitous data structures in modern applications, ranging from social networks, financial transactions and e-commerce platforms. For many practical scenarios, these graphs are often combined with temporal information, indicating dynamic characteristics, i.e., *temporal graphs*. For example, financial transactions have occurring timestamps; users' browsing records for products on e-commerce platforms are associated with timestamps as well. The dynamic patterns on graphs are of both research and practical interest, e.g., counting pattern's occurrences and predicting future interactions [1], [2].

We define a temporal graph as a graph with a timestamp sequence attached on its edges, representing times of occurred historical events between adjacent nodes [1]. In this paper, we study the problem of predicting *future* event time for adjacent nodes using the timestamp sequences of observed events between all adjacent nodes on a temporal graph. This problem can benefit many downstream applications. For example, user-item interaction networks are generally dynamic. By using the interaction graph information, recommendation systems may suggest proper items at the right moment to optimize its service quality and competitiveness [3]. Another example is traffic congestion prediction on a road network. Accurate traffic congestion time prediction for road segments helps navigation systems plan more intelligent routes [4]. The above applications necessitate predicting future event times precisely between nodes on temporal graphs.

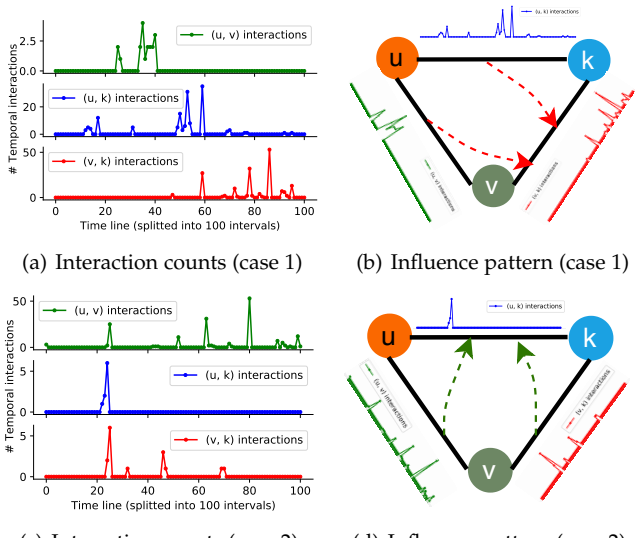
Recently, graph representation learning and graph neural networks (GNNs) [5] on static graphs [6], [7], [8], [9], [10] and on dynamic graphs [1], [11], [12], [13], [14], [15] have surged and become a popular tool to tackle prediction problems on dynamic graphs. However, these works can not be trivially adapted to the problem studied in this paper. Existing methods mainly focus their models on future link prediction [1], [11], [14], [16], [17] or future node state prediction tasks [1], [14], while the exact time prediction is rarely studied. Furthermore, predicting exact event times requires fine-grained representation of observed timestamps, while most graph learning models only take advantage of the chronological order of events. For example, some works partition dynamic graphs into ordered snapshots, and then embed each snapshot and combine them to obtain a dynamic representation [16], [18]. The coarse-grained processing discretizes the continuous temporal graph and eliminates exact timestamp information in each snapshot. Some other works adopt RNNs or temporal random walks to capture the sequential property [1], [12], [13], [19]. However, these methods only leverage

- *This work was done when visiting Singapore Management University.*
- *Wenwen Xia is with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: xiawenwen@sjtu.edu.cn*
- *Yuchen Li is with the School of Computing and Information Systems, Singapore Management University, Singapore 178902. E-mail: yuchenli@smu.edu.sg*
- *Shenghong Li is with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. He was also with the MoE Key Lab of Artificial Intelligence, AIInstitute, Shanghai Jiao Tong University, Shanghai 200240, China (Corresponding author). E-mail: shli@sjtu.edu.cn*

Manuscript received April 19, 2005; revised August 26, 2015.

1. In the following, we may use the term *temporal graph* and *dynamic graph* indiscriminately.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3149927, IEEE Transactions on Knowledge and Data Engineering



(a) Interaction counts (case 1) (b) Influence pattern (case 1)

(c) Interaction counts (case 2) (d) Influence pattern (case 2)

Fig. 1. We plot real interaction counts on two temporal triangle instance on the CollegeMsg dataset, and denote the three node pairs as (u, v), (u, k) and (v, k). We split the time into 100 intervals and count number of interactions at each interval. We can see that in Figure 1(a), interaction count of (v, k) increases gradually only after some earlier interactions on (u, v) and (u, k), which indicates that interactions on neighbor node pairs may stimulate interactions occurred on (v, k), depicted in Figure 1(b). Figure 1(c) and 1(d) are for the other sample, in which (u, v) and (v, k) may suppress interaction occurrence on (u, k).

the sequential order information of events as well, while the exact timestamp information and their correlations can not be modeled appropriately.

Another line of research for temporal events is closely related to stochastic processes, i.e., temporal point process [20]. Our work is inspired by the temporal point process framework because of its precise time modeling capability. Previous temporal point process models use either deterministic functions or learnable neural networks to define a *conditional intensity function* (CIF) to model a sequence of temporal events [21], [22], [23], [24], [25]. While these models regard each temporal sequence as independent to train model parameters. In our context of temporal graphs, these timestamp sequences are attached to edges, ans these edges are correlated via graphical structures. Training these temporal point process models with independent data samples may overlook the correlation information in the graph structure. For instance, comments from friends on social networks may have positive stimuli for a user's behavior. In contrast, activities from strangers may have much less influence. There exist some prior works considering both graphical structures and point processes [26], [27], [28]. However, most of these graph neural point process models adopt the Hawkes process or its variants and learn process parameters, which constraints their learning space and may induce inappropriate priors [29]. In contrast, we do not impose such priors and directly learn neural intensity functions to model neighbor's influences on targeted node pairs that we focus on.

In this paper, we propose a novel graph neural point process (GNPP) model to enable fine-grained prediction on temporal graphs. To utilize timestamp information precisely and build sequence correlations based on graph structures, we build GNPP by leveraging the message passing mechanism and the point process framework.

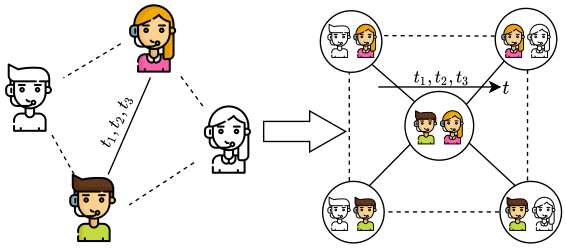


Fig. 2. From a message passing perspective, we treat each edge on a temporal graph as an *edge-node*. Hence temporal interaction information between node pairs are regarded as node features on the new *line graph*, facilitating the adoption of the message passing mechanism to model proximal neighbors' influence on the targeted node pair.

Note that our goal is to predict precise future interaction times for a given node pair, which may be influenced by neighborhood node pairs (two node pairs are called neighbors when they share one common node). Take temporal triangles on the CollegeMsg dataset as example, we plot two concrete temporal triangle instances in Figure 1. We plot interaction counts on these temporal edges w.r.t time in Figure 1(a) (case 1) and Figure 1(c) (case 2) respectively. For Figure 1(a), we can find that the interaction count of node pair (v, k) *increases* only after some previous interactions on its neighbor node pairs (u, v) and (u, k), indicating that neighbor node pairs' events have positive influences for the node pair (v, k), as depicted in Figure 1(b). While for case 2, the interaction count of (u, k) decreases after some interactions on (u, v) and (v, k), which may indicate suppression effects from neighbor node pairs. Therefore, it is important to capture these influence patterns, i.e., previous interactions' influence for the occurrence of interactions on their neighbor node pairs. Hence, we use the line graph of the original graph and regard the event prediction problem for node pairs as the time prediction for *nodes* on the line graph, as shown in the Figure 2. In this way, the node representations capture the interaction influences through the message passing mechanism [2].

For each node on the line graph, we construct its unique neural CIF function using its representation and a globally-shared module for modeling its dynamic pattern and conducting predictions. Each node's representation is computed considering its timestamp sequence and its neighbors' sequences. We adopt two self-attention modules to capture these two aspects respectively. Furthermore, to characterize timestamps in a fine-grained manner, we adopt a *time encoder* to map each real-valued timestamp to a high-dimensional vector. The position encoding proposed in [30] provides a reasonable but discrete scheme, making it inappropriate for continuous times. Instead, we reveal that two recently proposed time mapping schemes [14], [25] are mathematically equivalent with an analysis of their run-time difference. This saves us from tedious empirical evaluations between the two schemes. More importantly, it also provides insights for future works pertaining to time-sensitive prediction and reasoning tasks.

Our contributions include (1) modeling events occurrence patterns for temporal graph node pairs under the message passing and point process framework by GNPP.

2. If no specific instructions, in the following a node indicates one on the line graph, i.e., a node pair on the original graph.

(2) leveraging precise real-valued time embedding on temporal graphs and revealing the equivalence of two time encoding schemes; (3) conducting performance comparisons with several baselines, visualizing learned components, and discussing the optimization objectives in detail. The implementation of the GNPP model is released [3].

2 PRELIMINARIES

2.1 Temporal point process

Temporal point process is a widely adopted tool for modeling event sequences. Some seminal models include Hawkes process [31], self-correcting process [32] and Poisson process. The sequential data that these models focus on can be denoted as $\mathbf{x} = \{x_1, x_2, x_3, \cdots, x_{N_T}\}$. Each x_i represents a temporal event, which can be a simple real-valued time t_i in the single-variable case, or a tuple of time and event type (t_i, m_i) in the multi-variable case, where $m_i \in \mathcal{M}$ is the event type. $\mathcal{M}$ is the type space and is generally discrete.

The event distribution of a process model is characterized by the CIF $\lambda(t|\mathcal{H}_t)$ where $\mathcal{H}_t$ is the history (without loss of generality, we present the single-variable case here). Let $f(t_{n+1}|\mathcal{H}_{t_n})$ be the the conditional density function (CDF) of the next event time t_{n+1} and $F(t|\mathcal{H}_{t_n})$ be the corresponding cumulative distribution function (CUDF). Then the $\lambda(t|\mathcal{H}_t)$ is defined by

$$\lambda(t|\mathcal{H}_t) = \frac{f(t_{n+1}|\mathcal{H}_{t_n})}{1 - F(t|\mathcal{H}_{t_n})} \tag{1}$$

Further, $\lambda(t|\mathcal{H}_t)$ has a practical interpretation: for an infinitisemal interval dt, we have $\lambda(t|\mathcal{H}_t)dt = \mathbb{E}[N(t, t+dt)|\mathcal{H}_t]$, which is the expectation of number of events will occur during the region dt, and $N(A)$ is a counting measure of set $A \subseteq X$. The log-likelihood for an observed sequence $\mathbf{x} = \{x_i\}_{i=1}^{N_T}$ can be obtained by

$$l(\mathbf{x}) = \sum_{i=1}^{N_T} \log \lambda(t_i|\mathcal{H}_i) - \int_0^T \lambda(\tau|\mathcal{H}_\tau)d\tau \tag{2}$$

For instance, the seminal Hawkes process [31] defines its CIF as

$$\lambda(t_i|\mathcal{H}_i) = \mu + \alpha \sum_{t_i < t} \exp(-\beta(t - t_i)) \tag{3}$$

which means all past events have an exponentially decaying stimulus for future events' occurrence, combined with a constant base stimulus μ.

2.2 Message passing framework

GNNs utilize the message passing (MP) mechanism to capture the graph structure information [5], [33]. MP iteratively updates a node's representation by aggregating it's neighbors' representations. Generally each iteration captures 1-hop neighbors' information. Let $G = (V, E)$ denote a static graph where V is the node set and E is the edge set. For one layer's transformation, the MP process can be formulated as

$$\begin{aligned} \mathbf{a}_u^k &= \text{AGG}(\{\mathbf{h}_v^{k-1} | v \in N(u)\}) \\ \mathbf{h}_u^k &= \text{COMB}(\{\mathbf{h}_u^{k-1}, \mathbf{a}_u^k\}) \end{aligned} \tag{4}$$

3. https://github.com/xiawenwen49/GNPP-code.git.

The instantiations of AGG and COMB operations are critical and diverse for different GNNs. For instance, for the GCN model, $H^k = \tilde{A}H^{k-1}W^k$, in which AGG and COMB are integrated by the normalized Laplacian matrix $\tilde{A}$ [6]. While for the GAT model, the AGG is implemented using attention mechanism, and the COMB is composed of concatenation and linear transformation [7].

3 GNPP ARCHITECTURE

3.1 Problem formulation

Definition 1 (Temporal graph)**.** *A temporal graph is denoted by $G = (V, E, T)$, where V is the node set, E is the edge set and T is the timestamp set. For node i and node j, if $(i, j) \in E$, then $T_{ij} = \{t_1, \cdots, t_l\}$ represents l interaction/event timestamps on edge (i, j) where $t_i \in \mathbb{R}_{\geq 0}$. If $(i, j) \notin E$, then $T_{ij} = \emptyset$. We further define $T_{ii} = \{0\}$ for all node i. All these timestamps in T are currently observed and the maximum value is T_{max}.*

Definition 2 (The problem)**.** *Given a node pair (i, j) and the currently observed temporal graph G, we aim to predict the time of next interaction between node i and node j after T_{max}. We assume at least one timestamp has been observed on (i, j).*

3.2 Overview of GNPP

We present the framework of the GNPP model and the real-valued timestamp processing scheme in Figure 3. The time encoder is essentially a mapping that transforms each real-valued timestamp to a high-dimensional vector. Then GNPP will aggregate all these timestamp vectors from a node itself and its neighbors respectively. The aggregation is conducted using self-attention modules to capture neighborhood and ego influence patterns. Finally, we define a CIF based on the obtained representations and optimize model parameters using likelihood maximization under the temporal point process framework. In the below we will first present the forward computation layer of GNPP then the concrete timestamp processing and the optimization details.

3.3 Self-attention message passing

As illustrated in Figure 2, each node pair is regarded as a node, hence the following "nodes" refer to node pairs of the original temporal graph. GNPP considers the dependency between each node's event occurrence pattern with its neighbors. Following the message-passing framework, we aggregate event activities of neighbor nodes and combine the aggregated feature with the central node's activity feature to obtain high-level representations. We implement the aggregation operation using self-attention, and the aggregation can be naturally equipped with the "multi-heads" mechanism. We describe the computation details as follows.

We clarify that the space of event occurrence time is $\mathcal{X} := [0, T) \subset \mathbb{R}_{\geq 0}$. The timestamps set of node u is denoted by $\mathbf{x}_u$ and the set size is $|\mathbf{x}|$. For the k-th attention head, we first map a real-valued time t to a d-dimensional feature vector using a time encoder $\phi^{(k)}(t) : R_{\geq 0} \to \mathbb{R}^d$. Details of the time encoding function will be presented in Section 3.4. For

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3149927, IEEE Transactions on Knowledge and Data Engineering

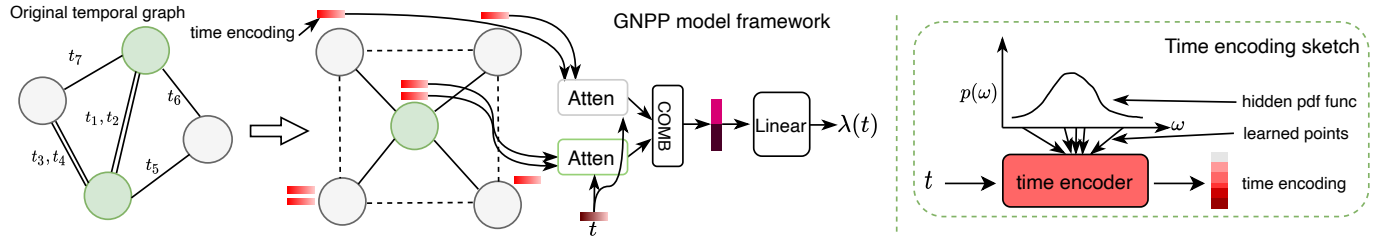


Fig. 3. The left part is the framework of the GNPP model. When operating on the line graph, each node will aggregate both its own historical timestamps and those from its neighbors, by two different self-attention modules. The learned representation support the subsequent CIF function and the exact time prediction. The right part is the sketch of the time encoding scheme. Some ω samples are optimized by gradient descent directly, acting like they are sampled from a hidden probability distribution.

attention head k, the aggregation and combination of the first layer for node u is presented as follows

$$\begin{aligned}
&\mathbf{a}_{u_{\text{neig}}}^{(k)}(t) = \text{AGG}_{\text{neig}}(\phi^{(k)}(t), \{\phi^{(k)}(t_{vj}) | t_{vj} \leq t, t_{vj} \in \mathbf{x}_v, v \in N(u)\}) \\
&\mathbf{a}_{u_{\text{self}}}^{(k)}(t) = \text{AGG}_{\text{self}}(\phi^{(k)}(t), \{\phi^{(k)}(t_{uj}) | t_{uj} \leq t, t_{uj} \in \mathbf{x}_u\}) \\
&\mathbf{h}_u^{(k)}(t) = \text{COMB}(\{\mathbf{a}_{u_{\text{neig}}}(t), \mathbf{a}_{u_{\text{self}}}(t)\})
\end{aligned} \tag{5}$$

For subsequent layer transformations, the $\phi^{(k)}(t)$, $\phi^{(k)}(t_{vj})$ and $\phi^{(k)}(t_{uj})$ in Eq (5) are just concatenated by former layer's $\mathbf{h}_u^{(k)}(t)$, $\mathbf{h}_v^{(k)}(t)$ and $\mathbf{h}_u^{(k)}(t)$ respectively.

We implement the $\text{AGG}_{\text{neig/self}}$ by the self-attention mechanism [30], which is computed as follows

$$\text{AGG}\{\phi(t), \{\phi(t_j)\}_{j=1}^n\} = \sum_{j=1}^{n} S(\frac{(Q\phi(t))^T K\phi(t_j)}{\sqrt{d^k}}) V\phi(t_j) \tag{6}$$

where S is the softmax layer, Q, K, V are learnable parameters and d^k is the input feature dimension. If $\mathbf{x}_u$ is empty, we compute Eq (6) by adding a 0 as the unique key. We implement the COMB function using a concatenation and a linear transformation as follows.

$$\text{COMB}\{\mathbf{a}_{u_{\text{neig}}}(t), \mathbf{a}_{u_{\text{self}}}(t)\} = W_c[\mathbf{a}_{u_{\text{neig}}}(t), \mathbf{a}_{u_{\text{self}}}(t)] \tag{7}$$

For multi-head attention which attends to information from different representation subspaces, we simply concatenate outputs from different heads.

$$\mathbf{h}_u(t) = [\mathbf{h}_u^{(0)}(t), \mathbf{h}_u^{(1)}(t), \cdots, \mathbf{h}_u^{(K-1)}(t)] \tag{8}$$

Based on the above definitions, we define the CIF for a single node u (a node pair we are interested in on the original temporal graph) as follows:

$$\lambda_u(t|\mathcal{H}_{u,t}) = g(\mathbf{w}^T \mathbf{h}_u(t) + b) \tag{9}$$

where $g(x) : \mathbb{R} \to \mathbb{R}^+$ is a monotonically increasing positive definite function to ensure the positiveness of conditional intensity function. As adopted in some previous works [22], [24], [25], we choose the softplus function $g(x) = \alpha \log(1 + e^{x/\alpha})$, which is a "smooth" version of the Relu function, where $\alpha > 0$ is an adjustable scalar.

3.4 Time encoding

In this section, we present the precise processing of real-valued timestamp information, i.e., time encoding. In our GNPP model, the temporal information is completely encoded by the time encoding function ϕ.

A most common instance for the time encoding $\phi(t)$ is *position encoding* [30], as indicated in Eq (10).

$$\phi_{pe}(i) := [\cdots, \sin(\frac{i}{10000^{2j/d}}), \cos(\frac{i}{10000^{2j/d}})]_{j=0}^{\frac{d}{2}} \tag{10}$$

where d is an even dimension number. While i represents an integer, which is the discrete order for encoding. Hence, position encoding can only handle discrete integers $i \in \mathbb{N}_0$. Though we can first discretize a continuous value and then encode it using the position encoding, we loss their continuous representations. More importantly, the (generalized) inner product structure $\phi(\cdot)^T Q K \phi(\cdot)$ equipped with fixed frequency $\phi_{pe}(\cdot)$ in Eq (6) can not model arbitrary influence patterns between timestamps, i.e., it is constrained by predefined position encoding parameters.

Recently, two time encoding schemes are proposed to model a general influence function considering the inner product structure of time encodings. One is the harmonic encoder [14] from the GNNs area and the other is the Fourier encoder [25] from the point process area. The main idea of these two encoding schemes is to construct a time mapping function $\phi(\cdot)$, such that a shift-invariant kernel function $\mathcal{K}(t_1, t_2) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, defined by the inner product of two time encodings $\mathcal{K}(t_1, t_2) = \psi(t_1 - t_2) = \phi(t_1)^T \phi(t_2)$ is positive semi-definite. In this way, the kernel function $\mathcal{K}(t_1, t_2)$ models the triggering effect of two events with time interval $t_1 - t_2$. We show that they are both based on the Bochner's theorem and are mathematically equivalent, to unveil their underlying properties and adopt an appropriate encoding scheme.

Theorem 1 (Bochner's theorem). *A continuous function $k(x)$ on $\mathbb{R}^d$ is positive semi-definite if and only if $k(x)$ is the Fourier transformation of a non-negative Borel measure $p(\omega)$.*

As stated in [34], if $\psi(\cdot)$ is properly scaled such that $\psi(0) = 1$, Bochner's theorem guarantees that $p(\omega) \geq 0$ and $\int_{\mathbb{R}^d} p(\omega) d\omega = 1$. In other words, we have $p(\omega)$ as a probability measure, and if we denote $\xi_\omega(x) = e^{j\omega x}$, the kernel function becomes an expectation of $\xi_\omega(x)\xi_\omega(y)^*$, as indicated in Eq (11).

$$\begin{aligned}
\mathcal{K}(x, y) = \psi(x - y) = \psi(y - x) &= \int_{\mathbb{R}^d} p(\omega) e^{-j\omega(y-x)} d\omega \\
&= \int_{\mathbb{R}^d} p(\omega) e^{j\omega(x-y)} d\omega = \mathbb{E}_{\omega \sim p(\omega)}[\xi_\omega(x)\xi_\omega(y)^*]
\end{aligned} \tag{11}$$

Note that a positive semi-definite function $\psi(\cdot)$ implicitly indicates that ψ is even, hence the equation $\psi(x-y) = \psi(y-x)$ holds and $p(\omega)$ is real. Since the $p(\cdot)$ and the $\mathcal{K}(\cdot,\cdot)$ are both real, the imaginary part of Fourier transform basis can be ignored and the following equation holds:

$$\begin{aligned}\mathcal{K}(t_1,t_2) &= \mathbb{E}_{\omega\sim p(\omega)}[\cos(\omega(t_1-t_2))] \\ &= \mathbb{E}_{\omega\sim p(\omega)}[\cos(\omega t_1)\cos(\omega t_2)+\sin(\omega t_1)\sin(\omega t_2)]\end{aligned} \tag{12}$$

The expectation in Eq (12) can be approximated by different ways. The harmonic encoder adopt the following encoding scheme

$$\phi_h(t) := \sqrt{\frac{1}{d}}[\cos(\omega_1 t), \sin(\omega_1 t), \cdots, \cos(\omega_d t), \sin(\omega_d t)] \tag{13}$$

where $\{\omega_i\}_{i=1}^d$ are learnable parameters. The Fourier encoder is

$$\phi_f(t) := \sqrt{\frac{1}{d}}[\sqrt{2}\cos(\omega_1 t + b_1), \cdots, \sqrt{2}\cos(\omega_d t + b_d)] \tag{14}$$

where $\{\omega_i\}_{i=1}^d$ are sampled from a neural network based distribution and $\{b_i\}_{i=1}^d$ are random values uniformly sampled from $[0, 2\pi]$.

We can show that they are both unbiased approximations of the expectation $\mathbb{E}_{\omega\sim p(\omega)}[\cos(\omega(t_1-t_2))]$. Since the b is uniformly sampled in $[0, 2\pi]$, we calculate the expectation of the inner product term $\phi_f(t_1)^T\phi_f(t_2)$ over b as

$$\begin{aligned}&\mathbb{E}_{b\sim[0,2\pi]}[2\cos(\omega t_1 + b)\cos(\omega t_2 + b)] \\ &\quad= \int_0^{2\pi}\frac{1}{2\pi}2\cos(\omega t_1 + b)\cos(\omega t_2 + b)db \\ &\quad= \int_0^{2\pi}\frac{1}{2\pi}[\cos(\omega(t_1-t_2)) + \cos(\omega(t_1+t_2)+2b)]db \\ &\quad= \cos(\omega(t_1-t_2))\end{aligned} \tag{15}$$

which indicates that $\phi_h(t_1)^T\phi_h(t_2)$ and $\phi_f(t_1)^T\phi_f(t_2)$ are equal taking expectation of ω over $p(\omega)$.

Revealing the equivalence helps to mitigate tedious evaluations on choosing an appropriate time encoding scheme for continuous timestamp representations. Moreover, the mathematical equivalence bridges the continuous time encoding methods in temporal graphs neural models and neural time series models. However, we note that there is a subtle difference between the two encoders. Since the distribution $p(\omega)$ determines the kernel function, the harmonic encoder proposes to directly learn ω samples, while the Fourier encoder firstly learns the distribution $p(\omega)$ and then samples points from the learned distribution. The subtle difference results in different parameters and learning efficiency. Besides, our experimental results in Section 4.7 reveal and verify the efficiency difference under both forward and backward computations. Considering that, we adopt the harmonic encoding $\phi_h(t)$ as timestamp representations, since it is more efficient and requires less learnable parameters. We show the sketch of this time encoding scheme in the right part of Figure 3.

3.5 Learning and inference

For all observed nodes and their associated event time sequences, we optimize the model using *maximum likelihood estimation* based on Eq (2). The log-likelihood in Eq (2) is computed by substituting $\lambda(t|\mathcal{H})$ with $\lambda_u(t|\mathcal{H}_{u,t})$ for a node u on the line graph with observed sequence $\mathcal{H}_u$. For the integral $\int_0^T \lambda(\tau|\mathcal{H}_\tau)d\tau$ in Eq 2, which is intractable for an analytic form. We adopt the Monte Carlo approach in Eq (16) for approximation [22], [24], [35].

$$\int_0^T \lambda(\tau|\mathcal{H}_\tau)d\tau \approx \frac{T}{K}\sum_{i=1}^K \lambda(\tau_i|\mathcal{H}_{\tau_i}), \quad \tau_i \sim U[0,T] \tag{16}$$

Once the $\lambda_u(t|\mathcal{H}_u)$ function is obtained, we can compute the conditional density function $f_u(t|\mathcal{H}_u)$ of next event time t_{n+1} according to Eq (1).

$$f_u(t|\mathcal{H}_u) = \lambda_u(t|\mathcal{H}_u)\exp\left(-\int_{t_n}^t \lambda_u(\tau|\mathcal{H}_u)d\tau\right) \tag{17}$$

And the prediction of t_{n+1} for node u is computed by

$$\hat{t}_{n+1} = \int_{t_n}^{+\infty} \tau f_u(\tau|\mathcal{H}_u)d\tau \tag{18}$$

In practice, computing the double integral for $\hat{t}_{n+1}$ by approximation leads to efficiency problem. Hence, we add another prediction module which takes $\mathbf{h}_u(t_n)$ as input and directly computes the prediction for next event time t_{n+1}, as in Eq (19). This prediction approach yields better computational efficiency and stability.

$$\hat{t}_{n+1} = \mathbf{w}_{\mathrm{p}}^T\mathbf{h}_u(t_n) + b_{\mathrm{p}} \tag{19}$$

We adopt the mean square error (MSE) to measure the prediction error by Eq (20).

$$l_{mse}(u) = \frac{1}{n_u - 1}\sum_{j=2}^{n_u}(\hat{t}_j - t_j)^2 \tag{20}$$

where n_u is the observed sequence length for node u. Note that we conduct $n_u - 1$ predictions for each timestamp sequence, ignoring the first record. We sum up the prediction error and the negative log-likelihood to obtain the final optimization objective.

$$L = \sum_{u\in G_{\text{line}}} -l(x_u) + \beta l_{mse}(u) \tag{21}$$

where β is an adjustable hyperparameter.

4 Experiments

In this section, we conduct experiments on three synthetic datasets and three real-world datasets. We compare our GNPP with seven graph neural models devised for temporal graphs and three recently proposed neural point process models. We evaluate these models using the rooted mean square error (RMSE) of predicted next event time and the log-likelihood (LL) value. Both metrics are widely adopted in previous works [22], [24], [25]. We compute the RMSE by predicting every held-out event time for every data sequence, i.e., for a sequence with length L, we make $L-1$ predictions according to histories until the latest observation time. For the settings of the GNPP model, we adopt 2 attention heads and 1-layer aggregation. This is because we experimentally observe that the 2-hop enclosing subgraph of a node on the line graph is generally sizeable and may

induce redundant information. The dimensions of the time encoder and the number of hidden units are set to 128, and the batch size is set to 1. We adopt the Adam optimization [36] and approximate the intergral in Eq (16) with 200 MC samples. We adopt 100 training epochs for GNPP. For all other parameters of baselines, we use default settings in their codes and original papers. More detailed settings could be found in the code released.

4.1 Baseline methods

In this section, we discuss baselines we used in experiments, including three neural point process models, i.e., Transformer Hawkes Process (THP) [24], Self-Attentive Hawkes Process (SAHP) [37], and Neural Network Point Process (NNPP) [38], and seven graph neural models, i.e., TGAT [14], TGN [13], GHNN [28], HTNE [26], DySAT [16], CTDNE [11], and GAT-LSTM. GAT-LSTM is a GAT model equipped with the harmonic encoder and an LSTM module as a naive GNN extended baseline, while others are originally devised for temporal graphs.

For three neural point process models, timestamp sequences for all observed temporal edges are directly feed into these models. For TGAT, GHNN, TGN, HTNE, DySAT and CTDNE, we construct node pairs like (u, v, t_i, t_{i+1}) from the original temporal graph. We extract representations of $r_u(t_i)$ and $r_v(t_i)$ from these learned models. Then we use a linear transformation to predict $\hat{t}_{i+1} = \mathbf{w}^T[r_u(t_i)|r_v(t_i)]$. For GAT-LSTM, the construction of the line graph is required. The timestamp sequence of each node (on the line graph) will firstly be encoded using a harmonic encoder, then the encoding sequence is passed through the LSTM module. The outputs of the LSTM module are regarded as node features and feed into the GAT model for the next event time prediction. Note that for THP, SAHP, NNPP, and our GNPP, we can evaluate both the LL and the RMSE. While for other models, we can only evaluate the RMSE since they can not produce LL values.

4.2 Datasets

We conduct evaluations on three synthetic datasets and three real-world datasets. For each dataset, we split node pairs with a 70%-15%-15% scheme for train, validation and test. For a node pair, only historical timestamps before its label timestamp (which we aims to predict) on itself and neighbor node pairs can be observed.

Synthetic datasets. The synthetic temporal graphs are generated by two steps. Firstly, we generate a random Watts–Strogatz small-world graph [39] $\bar{G} = (\bar{V}, \bar{E})$. We set all synthetic graphs to have 500 nodes and 1000 edges. Secondly, we simulate a point process with $|\bar{E}|$ event types, regarding each edge as an event type. For an edge (event type) e, the event occurred on its neighbors may has negative, positive, or null influence for its own future occurrences, i.e., making the interactions occur more frequently, rarely or as it is. Hence, we simulate a correlated effect between adjacent edges. For positive or negative influence, we adopt the commonly used Hawkes process to model the influence. For the independent case, we adopt the in-homogeneous Poisson process for all edges. The intensity function of an edge for all three cases can be summarized using Eq (22).

TABLE 1
The statistics of all datasets related to timestamps on graphs, including the average number of timestamps per edge (Average $|T_{ij}|$), the minimum and the maximum of timestamps.

Model	Average $\lvert T_{ij}\rvert$	Min t	Max t
Hawkes-negative	17.4	0	99.9
Hawkes-positive	23.5	0	99.9
In-homogeneous	61.7	0	89.7
Wikipedia	8.6	0	2678373.0
Reddit	2.3	0	99995.034
CollegeMsg	4.3	0	1098777142.0

$$\lambda_e(t) = \underbrace{\mu(t)}_{\text{base intensity}} + \underbrace{\alpha \sum_{\tau_n < t} \exp(-\beta_1(t-\tau_n))}_{\text{neighbor influence}} + \underbrace{\gamma \sum_{\tau_s < t} \exp(-\beta_2(t-\tau_s))}_{\text{self-influence}} \tag{22}$$

Note that τ_n represents a timestamp of an event that occurs on e's neighbor edges $N(e)$, while τ_s represents one that occurs on e itself. These three scenarios correspond to three parameter settings

1) Hawkes-negative, $\mu(t) = 1.0$, $\alpha = -0.2$, $\gamma = 0.5$, $\beta_1 = 2$, $\beta_2 = 2$.
2) Hawkes-positive, $\mu(t) = 0.1$, $\alpha = 0.5$, $\gamma = 0.1$, $\beta_1 = 2$, $\beta_2 = 2$.
3) In-homogeneous Poisson, $\mu(t) = \max(\frac{\sin(t)}{\sqrt{t+1}+0.1t}, 0.001)$, $\alpha = 0$, $\gamma = 0$.

Note the difference of parameter α in Hawkes-negative and Hawkes-positive, which simulates negative neighbor influence and positive one respectively. We conduct simulations in time range $[0, 100]$ using the Tick library [40]. Simulated time sequences are attached to corresponding edges on original graphs to obtain three temporal graphs.

Real-world datasets. We also conduct evaluations on three real-world temporal graphs.

Wikipedia is a temporal graph containing edited pages and users as nodes. Each edition between a user and an edge is recorded with a timestamp. This temporal graph contains about 9300 nodes and 160000 temporal edges. [1].

Reddit is an online community that records interactions between active users and their posts under subreddits. It has about 11000 nodes and 700000 temporal edges. [1]

CollegeMsg is a communication graph that records private Email interactions of an online society at the University of California, Irvine. This temporal graph contains about 2000 users and 60000 temporal edges [41].

In addition to the topological information, we also summarize timestamps-related statistics of all datasets in Table 1. Besides, since our model works on the line graph and large graphs have significantly more edges than nodes, constructing the line graph may be impractical because of memory and computation constraints. Considering the local property of GNNs, the line graph that GNPP operates on is not necessarily constructed completely. Small overlapped subgraphs centered on diverse nodes are sufficient.

TABLE 2
The RMSE comparison of all models on three synthetic temporal graphs (the smaller the better).

Model	Hawkes-neg	Hawkes-pos	Poisson
THP	24.61	21.79	7.54
SAHP	11.62	**2.86**	12.89
NNPP	59.26	14.90	9.86
GHNN	38.27	37.43	18.04
TGAT	35.96	33.51	3.19
TGN	12.01	1.36	12.51
HTNE	16.04	15.87	28.55
GAT-LSTM	85.52	92.84	85.40
DySAT	9.205	9.03	15.57
CTDNE	9.11	8.79	15.79
GNPP	**2.42**	8.21	**2.50**

TABLE 3
The LL comparison for neural point process models on three synthetic temporal graphs (the larger the better).

Model	Hawkes-neg	Hawkes-pos	Poisson
THP	-10.29	-3.99	**-0.31**
SAHP	-5.67	**-3.63**	-1.25
NNPP	-7.73	-4.38	-4.81
GNPP	**-4.37**	-4.60	-39.83

4.3 Synthetic datasets results

We first summarize the RMSE and LL in Table 2 and Table 3 respectively. The best and the second-best result are shown in bold and underlined respectively. For the RMSE metric, we can see that our method performs the best on Hawkes-neg and Poisson datasets and the second-best on Hawkes-pos dataset. For the LL metric, our method achieves the best on the Hawkes-neg dataset and performs relatively worse on Hawkes-pos and Poisson datasets. The reason is that Hawkes-pos and Poisson process are more consistent with the model form defined in THP and SAHP. The linear interpolation form in THP and the exponential form in SAHP make it more efficient to achieve a higher likelihood. Note that the RMSE target and the LL target are sometimes inconsistent. For instance, for the Poisson dataset, GNPP has a smaller (worse) likelihood in Table 3, while a better prediction error in Table 2. And we will discuss this inconsistency in detail in Section 5.

To understand the learned GNPP model more, we visualize the learned lambda scores on these synthetic datasets. The lambda score is the model's output before passed through the softplus function, i.e., the $\mathbf{w}^T\mathbf{h}_u(t) + b$ in Eq (9). To visualize the contributed score from the ego-attention module and the neighbor-attention module separately, we set the bias $b = 0$ and number of attention heads to 1 for both AGG_{neig} and AGG_{self}. Hence the lambda score contributed from AGG_{neig} and AGG_{self} can be computed by $\mathbf{w}[0:d]^T\mathbf{a}_{u_{\text{neig}}}(t)$ and $\mathbf{w}[d:2d]^T\mathbf{a}_{u_{\text{self}}}(t)$, respectively.

We choose 20 points in the time range $[0, 20]$ at equal intervals and obtain 20 time embeddings. The key, query, and value embeddings in Eq (6) are all set as these 20 embeddings. We compute $\mathbf{a}_{u_{\text{self}}}(t)$ and $\mathbf{a}_{u_{\text{neig}}}(t)$ by using the AGG_{self} and AGG_{neig} module on these time embeddings respectively. Then we can compute the lambda scores using the linear computation as stated above. Note that a key time embedding that occurs after a query time embedding will be ignored, and we put a mask in the attention module to meet this chronological constraint.

The lambda scores on the synthetic Hawkes-neg dataset are shown in Figure 4. We can see that the lambda scores computed by the self-attention module are positive, while scores computed by the neighbor-attention module are negative. Positive lambda scores mean that the learned model puts positive stimulus for an event that occurred on a node itself, while negative lambda scores reveal negative stimulus for an event that occurred on a neighbor node. The results in Figure 4 reveal that the GNPP model learned to pose negative influences on neighbor-events and positive influences on self-events.

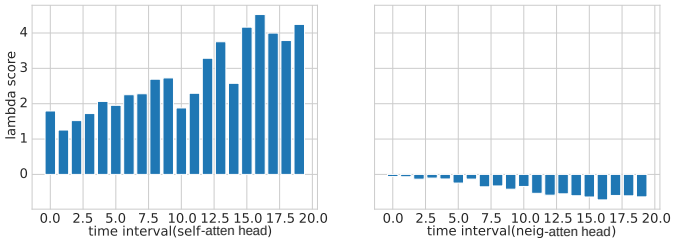


Fig. 4. Visualization of learned lambda scores for the synthetic hawkes-negative dataset. The horizontal axis represents the query time.

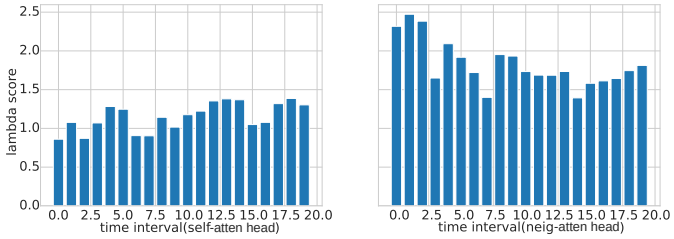


Fig. 5. Visualization of learned lambda scores for the synthetic hawkes-positive dataset. The horizontal axis represents the query time.

For the synthetic Hawkes-positive dataset, we can see in Figure 5 that the model poses positive lambda scores on both self-events and neighbor-events. This result shows the learned model believes both kinds of events have positive impacts on the occurrence of self-events. The result is also consistent with the synthetic setting for the Hawkes-positive dataset. In Figures 4 and 5, we can see that the GNPP model is capable of capturing positive or negative influence pattern from events occurred on neighborhood nodes.

4.4 Real-world datasets results

This section evaluates the performance of our method and baselines on the three real world temporal graphs. We report the RMSE results and LL results in Table 4 and Table 5 respectively. For all four graph neural baselines, we can find that TGAT performs better on the CollegeMsg, while CTDNE is better on the other two datasets. Our method achieves the best performance for LL on three real datasets. For the RMSE, we achieve the best on Wikipedia and CollegeMsg. The RMSE for Reddit is also relatively close to the best DySAT. This result indicates that the combination of self-attention mechanism and time encoding could capture temporal patterns on graphs more effectively and achieves better results. Further, we can see that SAHP achieves relatively considerable LL in Table 5, but it performs worse for RMSE in Table 4. This performance difference further shows that RMSE and LL are inherently inconsistent. Larger LL may not lead to smaller RMSE in some scenarios. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3149927, IEEE Transactions on Knowledge and Data Engineering

TABLE 4
The RMSE comparison of all models on real-world temporal graphs.

Model	Wikipedia	Reddit	CollegeMsg
THP	17.13	3.77	82.18
SAHP	361.07	427.20	540.62
NNPP	31.47	2.67	105.80
GHNN	116.26	4.47	240.47
TGAT	37.66	1.41	188.36
TGN	23.81	1.46	170.91
HTNE	12.72	1.61	665.95
GAT-LSTM	60.82	2.13	406.65
DySAT	22.79	0.86	318.51
CTDNE	22.51	**0.82**	320.78
GNPP	**11.14**	1.08	**29.75**

TABLE 5
The LL comparison for neural point process models on real-world temporal graphs.

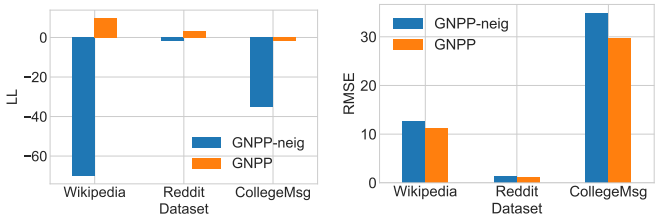
Model	Wikipedia	Reddit	CollegeMsg
THP	0.31	0.45	-1.56
SAHP	-1.07	-1.05	-2.32
NNPP	-5.40	-2.80	-6.10
GNPP	**9.85**	**3.50**	**-0.77**

phenomenon of this inconsistency is analogous to that in Section 4.3. We will discuss the LL-RMSE inconsistency here and that in Section 4.3 in Section 5.

4.5 Ignore neighbors VS Consider neighbors

In this section, we conduct an ablation study to evaluate the effectiveness of neighborhood information aggregation structure in our model on real-world cases. We compare performance between GNPP with neighbor attention module AGG_{neig} (denoted GNPP) and its counterpart without this module (denoted GNPP-neig) on three real-world datasets.

The results of LL and RMSE comparison are shown in Figure 6. We can see that for both metrics, the model with the neighbor attention module achieves a better performance, which indicates that the neighbor information aggregation mechanism benefits the temporal pattern capturing and future event time prediction. Besides, this aggregation mechanism has a larger influence on the LL metric than on the RMSE metric. We can see that in Figure 6(a), GNPP-neig has a much smaller likelihood than that of GNPP. While in Figure 6(b), the difference of RMSE between GNPP-neig and GNPP is relatively smaller than the difference of LL. The observed difference is due to $l_{mse}(u)$, which we introduced in the optimization objective in Eq (21). $l_{mse}(u)$ helps to narrow down the gap between GNPP and GNPP-neig for RMSE to some extent. Considering the results in Figure 6, we can conclude that the message aggregation



(a) LL comparison (b) RMSE comparison

Fig. 6. Evaluation of the GNPP model between with and without neighbor attention module.

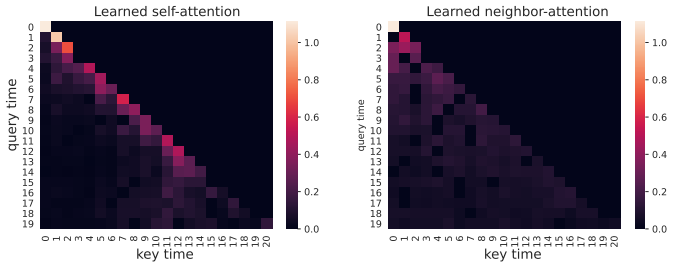


Fig. 7. Visualization of learned attention heads on the synthetic Hawkes-neg dataset.

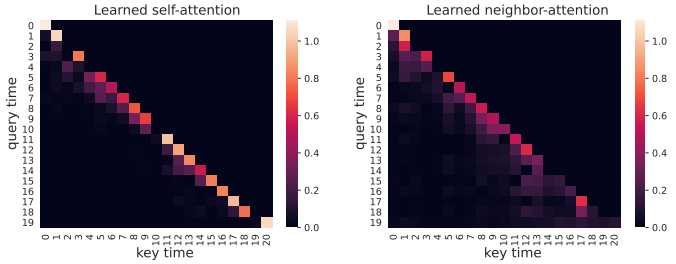


Fig. 8. Visualization of learned attention heads on the synthetic Hawkes-pos dataset.

from neighbors helps to promote model performance in modeling event patterns on temporal graphs.

4.6 Attention analysis

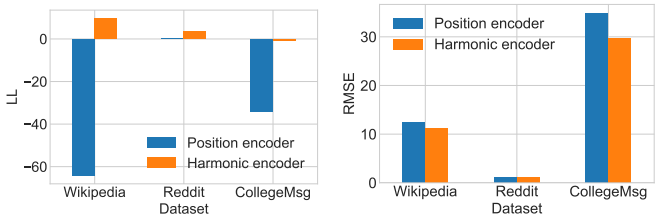
In this section, we visualize the learned attention weights on synthetic Hawkes-negative and Hawkes-positive datasets to gain more insights into the learned GNPP model. We construct 20 time encodings as keys and queries for attention weight computation. These time encodings are obtained in the same way as in Section 4.3. For each query encoding, we add masks on keys that are later than that query to adhere to the chronological constraint. We visualize both the self-attention module and the neighbor-attention module on the Hawkes-negative dataset in Figure 7. We can see that for the left figure in Figure 7, attention weights are large on the diagonal and decay fast for farther key encodings. While for the right neighbor-attention module, the weights are much smaller than the self-attention module and decay faster. This is because the likelihood optimization objective for the Hawkes-negative process forces the model to pay more attention to self-occurred events, since self-events stimulus more events while neighbor-events leads to less lambda summation terms in Eq (2).

Figure 8 visualizes attention results on the Hawkes-positive dataset. Both the self-attention and the neighbor-attention module pay more weights on key encodings which are proximal to the query encoding. Further, the weight decays quickly after a few grids, which means that the attention module captures the rapid decay influence for both self-events and neighbor-events. The decaying influence effect is analogous with that in Figure 7 because both Hawkes-positive and Hawkes-negative have exponentially decaying stimulus kernels, as formulated in Eq (22).

4.7 Performance comparison of time encoders

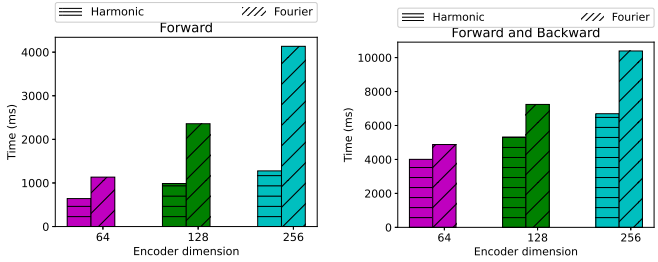
We have mentioned that there may have several different time encoding schemes to handle these real-valued times-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3149927, IEEE Transactions on Knowledge and Data Engineering



(a) LL comparison (b) RMSE comparison

Fig. 9. Comparison of LL and RMSE on three real-world datasets between the harmonic encoder and the position encoder.



(a) Forward (b) Forward and backward

Fig. 10. Efficiency comparison for harmonic encoder and Fourier encoder. The running time is accumulated over 10^4 timestamps. We illustrate both forward and backward computation times.

tamps, as in Section 3.4. In this section, we consider investigating the effects of these different time encoders, and the efficiency comparison between harmonic encoder and Fourier encoder. We mainly focus on the harmonic encoder and the position encoder. We conduct experiments on three real-world datasets and compare the LL and the RMSE metric for these two encoding schemes. Both encoding dimensions are set to 128, and other model settings are identical.

LL and the RMSE comparisons are shown in Figure 9(a) and Figure 9(b) respectively. The harmonic encoder outperforms the position encoder for both metrics, though the improvements differ across datasets. For the LL metric, the position encoder degrades the LL largely compared with the harmonic encoder, especially on the Wikipedia and the CollegeMsg dataset, as in Figure 9(a). The LL difference is relatively smaller on the Reddit dataset. We argue that this result comes from the difference of timestamp scales. The time scales for Wikipedia and CollegeMsg are much larger than that of Reddit. Time scales are reflected by the Max t shown in Table 1. For the RMSE metric, the difference is smaller on the Reddit dataset and larger on the Wikipedia and the CollegeMsg dataset. These results are concordant with those of the LL metric. Besides, we observe that the comparison results in Figure 9 resemble those in Figure 6, with some slight differences. This may reveal that both the neighborhood attention module and the harmonic encoder are important for a satisfactory model performance.

Figure 10 illustrates the efficiency comparison between harmonic encoder and Fourier encoder. We accumulate the computation time for 10^4 timestamps. We compare the forward computation time and forward+backward time. The results in Figure 10 show that for all three encoding settings, i.e., 64, 128, and 256 encoding dimensions, harmonic encoder has better efficiency than Fourier encoder. The results coincide with our analysis in Section 3.4. Moreover, the larger the encoding dimension, the larger the computational cost gap. Hence, our analysis in Section 3.4 provides a basis for selecting an appropriate encoding scheme with equivalent expressive power while better efficiency.

4.8 Visual insights of time encoders

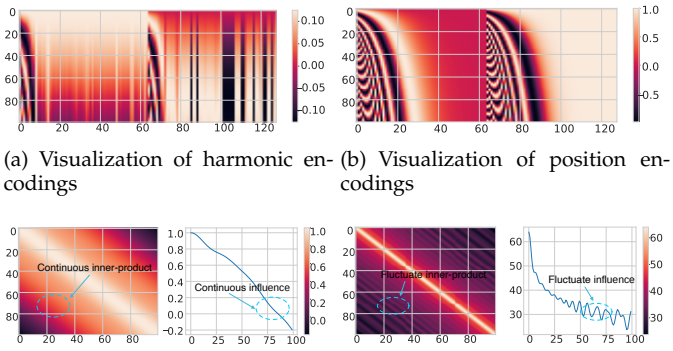
To understand the effects and functionalities of different time encoders more intuitively, we visualize the time encodings and their inner-products produced by the harmonic encoder and the position encoder in this section. The harmonic encoder is trained with the GNPP model on the Wikipedia dataset. For the position encoder, we replace the fixed number 10000 in Eq 10 with $1.1 \times \max(T)$ of Wikipedia dataset, for a more fine-grained discrete integer encoding. For both encoders, we fix the encoding dimension to 128.

Figure 11(a) and Figure 11(b) illustrate the images of time encodings produced by harmonic encoder and position encoder respectively. We select 100 time encodings from the time range $[0, 100]$, and the embeddings form a $100 * 128$ matrix, which is illustrated as an image. For both Figure 11(a) and Figure 11(b), the left part of the image presents the $\cos(\omega t)$ values, while the right part presents the $\sin(\omega t)$ values. Firstly, the scale of the learned harmonic encoder differs from that of the prefixed position encoder. The scale of learned harmonic may be caused by the timestamp scale of the specific dataset. While for position encoder, these time encodings are prefixed [4]. The fixed time encodings may lack flexibility for subsequent model learning and may induce worse prediction performance, compared with learnable time encodings. Secondly, we can observe that for the harmonic encoder, the time encodings show some specific and intriguing patterns, as for the right part in Figure 11(a). The encoding values are not as continuous as these of the position encoder, compared with the right pare of Figure 11(b). However, these intriguing patterns lead to continuous inner-products, and to some extent continuous influence functions, which we will discuss below.

Figure 11(c) and Figure 11(d) illustrate the inner-product matrix and the induced influence function induced by these time encodings. The left part of both Figure 11(c) and Figure 11(d) are computed using the equation $Img = \phi(T) \times \phi(T)^T$. $\phi(T) \in \mathbb{R}^{100 \times 128}$ is the time encoding matrix, and $Img \in \mathbb{R}^{100 \times 100}$ represents the inner-product matrix of time encodings. The right parts of Figure 11(c) and Figure 11(d) plot the first line of their corresponding Image matrix, indicating the inner-product induced influence w.r.t time intervals (the interval ranges from 0 to 100). Comparing Figure 11(c) and Figure 11(d), we can observe that the learned harmonic encoder produces time embeddings which result in continuous inner-product values, while the position encoder leads to fluctuating ones. The right parts of Figure 11(c) and Figure 11(d) illustrate this effect more clearly, as we point out with blue dotted ellipses. The fluctuate curve in the right part of Figure 11(d) comes from prefixed coefficients in position encodings. To some extent, inner-product values of time encodings indicate influence stimulus intensities.

4. Although we can set different maximum time values, i.e., the denominator in Eq (10), for different datasets, the time encodings are frozen for the same dataset.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3149927, IEEE Transactions on Knowledge and Data Engineering



(a) Visualization of harmonic encodings (b) Visualization of position encodings

(c) Visualization of harmonic encoder inner-products (d) Visualization of position encoder inner-products

Fig. 11. Visualization of harmonic and position encodings, and visualization of two inner-product induced stimulus functions.

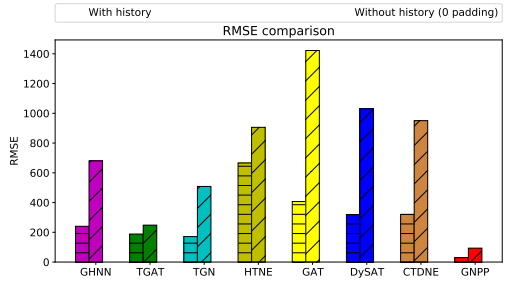


Fig. 12. Comparison of 0-padding between GNPP and the baselines.

Hence, this difference supports the conclusion that the learnable harmonic encoder is more suitable for capturing the continuous influence effect on real-world temporal graphs. While predefined position encodings lack flexibility and increase time encoding induced biases.

4.9 The effect of padding 0 for fresh node pairs

In this section, we discuss effects of 0-padding for node pairs without historical timestamps observed. Since point process baselines require historical timestamps to predict future ones, they are not eligible for this comparison. We compare GNPP with those baselines considering the graph structure. We first select a threshold as $T_{thr} = 0.7 \times T_{max}$. For node pairs whose first timestamps are larger than T_{ths}, we select them as test cases and predict their first timestamp, with other timestamps removed and a 0 padded. The first timestamp is regarded as the label for all these node pairs. We conduct experiments on the CollegeMsg dataset and illustrate results in Figure 12. We can see that all methods have larger RMSE errors than their original settings. However, GNPP still holds the smallest error compared with other baselines . We suggest that the padding trick is a simple and applicable solution for node pairs without interaction history. More advanced methods can be investigated in future works.

5 Discussion and Limitations

In this section, we discuss two interesting aspects of our problem and limitations of our solution.

The first aspect is the objective function. Many previous temporal point process works adopt the (negative log) likelihood as a whole or an important part of the optimization objective. From prediction results in Section 4.3 and Section 4.4, we observe that a larger likelihood may not lead to a smaller prediction error. For example, GNPP has a worse LL in Table 3 with a better RMSE in Table 2, and SAHP has a better LL in Table 5 with a worse RMSE in Table 4. We argue that the reason comes from the inconsistency between likelihoods and prediction errors. Investigating the likelihood in Eq (2), we can find that it consists of two parts, i.e., the (log) intensity summation on occurred events $\sum_{i=1}^{N_T} \log \lambda(t_i|\mathcal{H}_i)$, and the negative integral in the whole period $-\int_0^T \lambda(\tau|\mathcal{H}_\tau)d\tau$. Imaging that a neural network model aims to achieve a large likelihood value, it may learn to increase the intensity function values on occurred events, and decrease the intensity value drastically during the interval of two occurred events. In this way, the neural model may resemble an impulse function on occurred event times. However, this kind of neural intensity function impairs its ability to predict the future, i.e., increasing prediction error. This circumstance may happen if the model has a high capacity (roughly speaking, many parameters) and data sequences are insufficient. Hence, we argue that for this kind of neural point process models (include ours), either adjusting the objective function or constraining the model complexity requires careful consideration.

The second aspect we discuss is the relation between our model and previous representation learning methods for dynamic or temporal graphs. In our model, we induce a CIF function based on high-level node representations, and these representations aggregate both topological and temporal information of the temporal graph. From this perspective, the model can also be regarded as a more general edge representation learning model on temporal graphs, as previous representation learning works [11], [16], [19]. The whole framework may support other prediction tasks on temporal graphs, by stacking other neural architectures on the output representations in Eq (8) with fine-tuning. However, the generalization ability of representations learned by the GNPP model requires more verification on other tasks on temporal graphs, and we may leave this as future works.

There are also some limitations for the proposed model. Firstly, in the message aggregation process, we do not distinguish different event types in different neighborhoods. Instead, we regard all events as homogeneous for all neighbor edges. While in practice, there may have diverse event types. Secondly, in this work, we model the temporal graph as a graph with merely timestamp information without node and edge features. Because many temporal graph datasets have no auxiliary features. This abstraction may be over-simplified and integrating these rich features into model computation appropriately to improve task-specific performance deserves more consideration.

6 Related Work

Our work is closely related to two research fields: (1) Temporal point process; (2) Graph neural networks.

Temporal point process. Classic temporal point process models assume pre-defined CIF forms [3], [42]. The CIF function is used to model future event occurrence densities based on currently observed history. These models benefit from well interpretability and learning efficiency

while suffers constrained capability of modeling complex dynamics. The success of deep neural networks motivates some works to introduce neural networks into the temporal point process framework [21], [22], [23], [24], [25], [43]. [21], [22] propose to utilize RNNs to capture the sequential information of temporal events and define CIF functions. [23] adopts reinforcement learning (RL) to model the observed event sequences. [24], [25] both adopt the attention mechanism to model previous events' influence on future events' occurrence and to define CIF functions. There are also some works considering both structural correlations and event histories. [27] models each temporal sequence using a Hawkes process and sequence correlations using a GCN with graph regularizers. [28] focuses on knowledge graphs and combines the Hawkes process and neighbor information aggregation to conduct future entity or event time predictions. [44] jointly models both intra-series and inter-series correlations using GFT and DFT, and adopts a neural network to learn the patterns lying in the spectral domain of the GFT-transformed graph signals. However, most of these graph neural point process models consider the Hawkes process or variants, e.g., [27], [28], [29], [45], [46]. In contrast, GNPP directly models the CIF function, instead of imposing such point process model priors.

Graph neural networks. Recently, GNNs have achieved noticeable improvements on some graph-related tasks, e.g., link prediction [47], [48], [49], node classification [6], [7] and graph classification [50], [51]. Some seminal works including GCN [6], GAT [7] and others [8], [10], [14], [52] have promoted researches that utilizing neural networks to extract nodes' representations based on graph structures. Recently, many works devote to extending GNNs on temporal graphs [1], [11], [12], [53]. DyRep focuses on modeling two types of events on temporal graphs using two correlated temporal point processes. These two processes are based on node representations aggregated by GNN layers [12]. TGAT utilizes time encoding and self-attention to conduct temporal message-passing layers [14]. TGN [13] devises an embedding module and a memory module to memorize each achieved event and update the embeddings of corresponding nodes. HTNE [26], CTDNE [11], JODIE [1] focus on embedding learning on a temporal graph, by integrating point processes, random walks or RNN modules. Note that JODIE is originally devised for bipartite graphs, and it is non-trivial to extend it to general temporal graphs. DySAT aggregates node representations using self-attention and captures temporal information using position encoding and another self-attention pipeline [16]. Our work differs from previous ones by capturing precise real-valued timestamp information using a mapping function and directly modeling node pair relations.

7 Conclusion

We have presented the Graph Neural Point Process model that aims to model the temporal event occurrence patterns on graphs and predict future interaction times. GNPP captures precise real-valued timestamp information by high-dimensional vector mapping and utilizes the message-passing mechanism to obtain both spatial and temporal information. The representations form the basis of defining conditional intensity functions in the temporal point process framework. In empirical evaluations on synthetic and real-world datasets, GNPP shows better likelihood value and event prediction error compared with several recently proposed neural point process models and graph neural models suitable for temporal graphs. Meanwhile, we discuss the influence of commonly adopted objective functions in temporal prediction tasks. In this work we treat all neighbor temporal edges indiscriminately, i.e., we regard them as homogeneous. In the future, we aim to extend the model to handle the heterogeneity and to utilize node and(or) edge features in model computation.

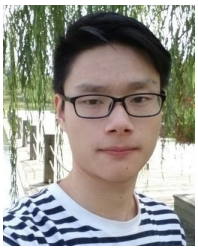
Acknowledgments

This research work is funded by the National Nature Science Foundation of China under Grant 61971283 and U20B2072, and Shanghai Municipal Science and Technology Major Project under Grant 2021SHZDZX0102. Yuchen Li's work is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No.: MOE2019-T2-2-065). Wenwen Xia's work is also funded by the China Scholarship Council.

References

[1] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *KDD*, 2019.
[2] J. Wang, Y. Wang, W. Jiang, Y. Li, and K.-L. Tan, "Efficient sampling algorithms for approximate temporal motif counting," in *CIKM*, 2020.
[3] N. Du, Y. Wang, N. He, and L. Song, "Time-sensitive recommendation from recurrent user activities," in *NIPS*, 2015.
[4] Z. Diao, X. Wang, D. Zhang, Y. Liu, K. Xie, and S. He, "Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting," in *AAAI*, 2019.
[5] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.
[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2016.
[7] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.
[8] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *NIPS*, 2017.
[9] W. Xia, Y. Li, J. Wu, and S. Li, "DeepIS: Susceptibility estimation on social networks," in *WSDM*, 2021.
[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *ICLR*, 2019.
[11] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *WWW*, 2018.
[12] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "DyRep: Learning representations over dynamic graphs," in *ICLR*, 2019.
[13] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," in *ICML*, 2020.
[14] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.
[15] W. Xia, Y. Li, J. Tian, and S. Li, "Forecasting interaction order on temporal graphs," in *KDD*, 2021.
[16] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "DYSAT: Deep neural representation learning on dynamic graphs via self-attention networks," in *WSDM*, 2020.
[17] A. García-Durán, S. Dumančić, and M. Niepert, "Learning sequence encoders for temporal knowledge graph completion," in *EMNLP*, 2020.
[18] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–36, 2014.

[19] X. Chang, X. Liu, J. Wen, S. Li, Y. Fang, L. Song, and Y. Qi, "Continuous-time dynamic graph learning via neural interaction processes," in *CIKM*, 2020.
[20] J. A. González, F. J. Rodriguez-Cortes, O. Cronie, and J. Mateu, "Spatio-temporal point process statistics: a review," *Spatial Statistics*, vol. 18, pp. 505–544, 2016.
[21] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *KDD*, 2016.
[22] H. Mei and J. Eisner, "The neural hawkes process: A neurally self-modulating multivariate point process," in *NIPS*, 2017.
[23] S. Li, S. Xiao, S. Zhu, N. Du, Y. Xie, and L. Song, "Learning temporal point processes via reinforcement learning," in *NIPS*, 2018.
[24] S. Zuo, H. Jiang, Z. Li, T. Zhao, and H. Zha, "Transformer hawkes process," in *ICML*, 2020.
[25] S. Zhu, M. Zhang, R. Ding, and Y. Xie, "Deep fourier kernel for self-attentive point processes," in *AISTATS*, 2020.
[26] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," in *KDD*, 2018.
[27] J. Shang and M. Sun, "Geometric hawkes processes with graph convolutional recurrent neural networks," in *AAAI*, 2019.
[28] Z. Han, Y. Ma, Y. Wang, S. Günnemann, and V. Tresp, "Graph hawkes neural network for forecasting on temporal knowledge graphs," in *AKBC*, 2020.
[29] Q. Zhang, A. Lipani, and E. Yilmaz, "Learning neural point processes with latent graphs," in *WWW*, 2021.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.
[31] A. G. Hawkes, "Spectra of some self-exciting and mutually exciting point processes," *Biometrika*, vol. 58, no. 1, pp. 83–90, 1971.
[32] V. Isham and M. Westcott, "A self-correcting point process," *Stochastic Processes and their Applications*, vol. 8, no. 3, pp. 335–347, 1979.
[33] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, 2008.
[34] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS*, 2007.
[35] C. Robert and G. Casella, *Monte carlo statistical methods*. Springer Science & Business Media, 2013.
[36] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
[37] Q. Zhang, A. Lipani, O. Kirnap, and E. Yilmaz, "Self-attentive hawkes processes," in *ICML*, 2020.
[38] T. Omi, N. Ueda, and K. Aihara, "Fully neural network based model for general temporal point processes," in *NIPS*, 2019.
[39] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[40] E. Bacry, M. Bompaire, S. Gaïffas, and S. Poulsen, "Tick: a Python library for statistical learning, with a particular emphasis on time-dependent modeling," *arXiv*, 2017.
[41] P. Panzarasa, T. Opsahl, and K. M. Carley, "Patterns and dynamics of users' behavior and interaction: Network analysis of an online community," *Journal of the American Society for Information Science and Technology*, vol. 60, no. 5, pp. 911–932, 2009.
[42] B. Yuan, H. Li, A. L. Bertozzi, P. J. Brantingham, and M. A. Porter, "Multivariate spatiotemporal hawkes processes and network reconstruction," *SIAM Journal on Mathematics of Data Science*, vol. 1, no. 2, pp. 356–382, 2019.
[43] U. Upadhyay, A. De, and M. Gomez-Rodriguez, "Deep reinforcement learning of marked temporal point processes," in *NIPS*, 2018.
[44] D. Cao, Y. Wang, J. Duan, C. Zhang, X. Zhu, C. Huang, Y. Tong, B. Xu, J. Bai, J. Tong *et al.*, "Spectral temporal graph neural network for multivariate time-series forecasting," in *NIPS*, 2020.
[45] T. Li, T. Luo, Y. Ke, and S. J. Pan, "Mitigating performance saturation in neural marked point processes: architectures and loss functions," in *KDD*, 2021.
[46] M. Yao, S. Zhao, S. Sahebi, and R. F. Behnagh, "Relaxed clustered hawkes process for student procrastination modeling in moocs," in *AAAI*, 2021.
[47] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *NIPS*, 2018.
[48] M. Zhang, Z. Cui, S. Jiang, and Y. Chen, "Beyond link prediction: Predicting hyperlinks in adjacency space," in *AAAI*, 2018.
[49] L. Cai, J. Li, J. Wang, and S. Ji, "Line graph neural networks for link prediction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.
[50] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of Computer-Aided Molecular Design*, vol. 30, no. 8, pp. 595–608, 2016.
[51] D. K. Johnson and J. Karanicolas, "Ultra-High-Throughput Structure-Based Virtual Screening for Small-Molecule Inhibitors of Protein-Protein Interactions," *Journal of Chemical Information and Modeling*, vol. 56, no. 2, pp. 399–411, 2016.
[52] R. Liao, Z. Zhao, R. Urtasun, and R. S.Zemel, "LanczosNet: Multi-scale deep graph convolutional networks," in *ICLR*, 2019.
[53] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dynamic graph representation learning via self-attention networks," in *ICLRW*, 2019.



Wenwen Xia is a Ph.D. candidate with the school of electrical information and electric engineering, Shanghai Jiao Tong University. He is currently visiting the School of Computing and Information Systems, Singapore Management University. He received the B.S. degree in information security from the school of computer science, Wuhan University. His research interests mainly focus on novel applications of Graph Neural Networks and their theoretical expressive power. He published several papers on some data mining conferences, including KDD, WSDM, ICDM, etc.

Yuchen Li is an assistant professor with the School of Computing and Information Systems, Singapore Management University (SMU). He received double BSc degrees in applied math and computer science (both with first-class honors) and a Ph.D. degree in computer science from the National University of Singapore (NUS), in 2013 and 2016, respectively. His research interests include graph analytics and heterogeneous computing. He published in top database and data mining venues such as SIGMOD, VLDB, ICDE, KDD, and TheWebConf. He received best paper awards in KDD and a Lee Kong Chian fellowship.

Shenghong Li received the B.S. and M.S. degrees in electrical engineering from the Jilin University of Technology, Jilin, China, in 1993 and 1996, respectively, and the Ph.D. degree in radio engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 1999. Since 1999, he has been a Research Fellow, an Associate Professor, and a Professor with Shanghai Jiao Tong University, Shanghai, China. In 2010, he was a Visiting Scholar with Nanyang Technological University, Singapore. He is currently a Professor with the School of Cyber Space Security, Shanghai Jiao Tong University. His research interests include information security, signal and information processing, and artificial intelligence.