Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank

Yiming Li
Department of CSE, HKUST
yliix@cse.ust.hk

Yanyan Shen
Department of CSE, Shanghai Jiao Tong University
shenyy@sjtu.edu.cn

Lei Chen
Department of CSE, HKUST
DSA Thrust, HKUST (GZ)
leichen@cse.ust.hk

Mingxuan Yuan
Huawei Noah's Ark Lab
Yuan.Mingxuan@huawei.com

ABSTRACT

Temporal graph neural networks (T-GNNs) are state-of-the-art methods for learning representations over dynamic graphs. Despite the superior performance, T-GNNs still suffer from high computational complexity caused by the tedious recursive temporal message passing scheme, which hinders their applicability to large dynamic graphs. To address the problem, we build the theoretical connection between the temporal message passing scheme adopted by T-GNNs and the temporal random walk process on dynamic graphs. Our theoretical analysis indicates that it would be possible to select a few influential temporal neighbors to compute a target node's representation without compromising the predictive performance. Based on this finding, we propose to utilize T-PPR, a parameterized metric for estimating the influence score of nodes on evolving graphs. We further develop an efficient single-scan algorithm to answer the top-k T-PPR query with rigorous approximation guarantees. Finally, we present Zebra, a scalable framework that accelerates the computation of T-GNN by directly aggregating the features of the most prominent temporal neighbors returned by the top-k T-PPR query. Extensive experiments have validated that Zebra can be up to two orders of magnitude faster than the state-of-the-art T-GNNs while attaining better performance.

PVLDB Reference Format:
Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. PVLDB, 16(6): 1332 - 1345, 2023.
doi:10.14778/3583140.3583150

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/LuckyLYM/Zebra and https://gitee.com/mindspore/models/tree/master/research/gnn/zebra.

1 INTRODUCTION

Many real-world applications rely on dynamic graphs, where nodes and edges are constantly updated. For instance, users on social platforms like Reddit interact with each other by posting a comment; users on an e-commerce platform interact with items by making a purchase. The above applications require learning representations of nodes upon new interactions to support a wide range of downstream tasks, including temporal link prediction and dynamic node classification. Temporal graph neural networks (T-GNNs) [22, 36, 49, 50, 54] have achieved state-of-the-art performance in learning representations on dynamic graphs. Given a target node at a specific timestamp, T-GNNs compute its temporal embedding through *recursive temporal message passing*, which consists of time-dependent neighbor sampling and time-encoded neighborhood aggregation. The above time-related modules help T-GNNs to better capture the diffusion process on evolving graphs. Recent research [36, 50] shows that T-GNNs can significantly outperform the static GNNs [17, 18, 42] and snapshot-based GNNs [15, 16, 33] in predictive performance.

Unfortunately, the high time complexity of T-GNNs in both training and inference stages hinders their applicability to large dynamic graphs. As shown in Figure 1a, the computation graph of T-GNN grows exponentially with the model depth, which is known as the *neighbor explosion* problem. This issue could cause severe efficiency and memory concerns. For instance, CAW [50] fails to finish one epoch of model training on the Wiki-Talk (7.8M edges and 1.14M nodes) dataset [4] with an RTX 2080 Ti GPU in 12 hours; TGN [36] suffers from slow inference on large dynamic graphs due to the excessive memory consumption caused by recursive temporal neighborhood aggregation. To the best of our knowledge, improving the efficiency of T-GNNs still remains an open problem to be resolved.

In this work, our ultimate goal is to reduce the computational cost of T-GNNs without compromising model performance. To this end, we get inspiration from the well-acknowledged fact [19] that not all neighbors are equally important for learning node embeddings on static graphs. Intuitively, given a target node v, neighbors more densely connected to v are supposed to make a larger impact in learning its representation. Therefore, prior works [8, 19, 20, 43] resort to graph diffusion like personalized PageRank (PPR), heat kernel PageRank, and Katz index for calculating the proximity score between nodes on static graphs. These methods then select and aggregate the most influential neighbors to compute a target node's embedding. Particularly, the diffusion-based GNNs [8, 19] can be orders of magnitude faster than the vanilla GNNs [17, 42] while achieving higher model accuracy. This is because the diffusion-based approaches can eliminate the noises introduced by insignificant neighbors and thus resolve the over-smoothing [10, 24, 31] issue caused by increasing model depth.

Based on the above discussion, we raise up a natural question: *do there exist influential nodes that we can utilize to accelerate the*

Proceedings of the VLDB Endowment, Vol. 16, No. 6 ISSN 2150-8097.
doi:10.14778/3583140.3583150

computation of T-GNNs without compromising model accuracy, just as the efforts devoted to static GNNs?

To answer this question, we analyze how much each input feature affects the final output of T-GNNs using influence functions [21, 55, 59]. Specifically, T-GNN learns a weight for each selected neighbor to compute a target node's embedding. Our key observation is that the *weights used for temporal message passing* in T-GNNs correspond to the *1-step temporal random walk probabilities* on dynamic graphs. We theoretically prove that the influence of input node x on T-GNN's output embedding of node y is proportional to the probability that a temporal random walk starting from node y terminates at node x. Intuitively, neighbors with large reaching probabilities tend to be more crucial in learning a node's embedding. Thus, not all neighbors contribute equally to T-GNN computation. Based on the analysis, we give a positive answer to the previous question. By distinguishing and aggregating the most influential neighbors, we can possibly reduce the computational cost of T-GNNs without sacrificing model accuracy.

To compute the influence score of nodes on evolving graphs, we need to estimate the temporal random walk probabilities based on neighbor weights learned by T-GNNs. The challenge here is that we cannot know the neighbor weights in advance without running T-GNN. Fortunately, we find that given a target node, the weights of its top-10 recently interacted neighbors learned by vanilla T-GNNs approximately follow exponential distributions (see details in Section 4.3). This is reasonable since recent interactions capture the latest trend, thus conveying more instructive information in the diffusion process on dynamic graphs. Following the observation, we propose temporal Personalized PageRank (T-PPR), a novel parameterized metric for estimating the influence score of nodes on dynamic graphs. Specifically, T-PPR is configured by an exponential decay factor β ($0 < \beta < 1$), and for a target node y, T-PPR assigns its i-th recently interacted neighbor transition probability proportional to β^i. Then, T-PPR computes the influence score of a node x on the target node y as the temporal random walk probability from y to x. Therefore, we can identify the top-k influential neighbors by solving a top-k T-PPR query.

In this work, we are the first to extend PPR to T-PPR and establish the theoretical connection between T-GNN and T-PPR. We further develop Zebra, a scalable framework that improves T-GNN via T-PPR. In general, Zebra mimics the recursive temporal neighborhood aggregation operation of T-GNNs through answering top-k T-PPR queries, thus eliminating expensive message aggregation. Specifically, given a target node v at a specific timestamp t, Zebra first searches the most informative temporal neighbors with respect to (v, t) by answering the top-k T-PPR query. Then, Zebra aggregates the features of selected neighbors with normalized T-PPR values. In this way, Zebra resolves the neighbor explosion issue, thus improving the computational efficiency of T-GNNs.

The core technical challenge of Zebra is to answer T-PPR queries efficiently. Otherwise, the cost saved from avoiding recursive temporal neighborhood aggregation could be over-consumed by the query algorithm itself. Although many advanced techniques [26, 28, 46–48, 52] have been developed for PPR query processing, they cannot be adapted to efficiently answer top-k T-PPR queries due to the *highly imbalanced weight distribution*. Specifically, existing PPR solutions usually assume that the input static graph is unweighted and cyclic, while T-PPR handles highly skewed and acyclic graphs. For weighted graphs, the push-based [6, 7, 29, 46, 47, 57] and iteration-based [26, 32, 52] PPR methods would spend significant time on calculating a tiny probability mass, resulting in severe overhead. To address this challenge, we leverage the acyclic property of T-PPR to build a single-scan near-linear algorithm that can solve the top-k T-PPR query upon new interactions with approximation guarantees. Besides, T-PPR metrics configured by different exponential decay factors have distinct characteristics. A single T-PPR, if not well parameterized, may fail to sufficiently capture the diffusion process on dynamic graphs, thus degrading model accuracy. To make Zebra more stable and performant, we propose to build robust T-GNN by constructing an ensemble of temporal embeddings obtained with different T-PPR metrics.

In summary, we have made the following contributions:

- We build the theoretical connection between the temporal message passing scheme in T-GNNs and the temporal random walk on dynamic graphs. Our theoretical finding shows that the influence of an input node x on T-GNN's output node y is proportional to the temporal random walk probability from y to x.
- We propose to utilize T-PPR, a novel parameterized metric for estimating the influence score of nodes on evolving graphs. Computing exact T-PPR values is computationally prohibitive. Thus, we propose an efficient single-scan algorithm to answer the top-k T-PPR query with rigorous approximation guarantees.
- We present Zebra, a scalable framework for accelerating the training and inference of T-GNN based on T-PPR. Zebra avoids the cumbersome message passing process required by vanilla T-GNN through directly aggregating the most prominent neighbors returned by the top-k T-PPR query.
- We have conducted extensive experiments to validate the efficiency and effectiveness of Zebra. Specifically, Zebra can be up to two orders of magnitude faster than the state-of-the-art T-GNNs while attaining better predictive performance on various dynamic graphs.

Overview. In the rest of the paper, we introduce our technical background in Section 3, investigate the connection between T-PPR and T-GNN in Section 4, and present our framework Zebra in Section 5. Then, Section 6 introduces our solution to the top-k T-PPR query. Finally, we show experiments in Section 7, review related work in Section 2, and conclude this paper in Section 8.

2 RELATED WORK

Temporal Graph Neural Networks. T-GNNs are powerful models for learning representations on continuous-time dynamic graphs. The pioneering work [22] uses a time projection module to predict future embeddings and adopts RNNs to update state vectors of nodes. DyRep [41] further incorporates 2-hop neighborhood when updating node representations. TGAT [54] mimics the message passing scheme of static GNNs and encodes time information through random Fourier features. TGN [36] unifies previous methods [22, 41, 54] and have achieved remarkable accuracy improvement. CAW [50] encodes sampled random walks using an anonymization strategy to further improve the predictive performance on unseen nodes. Moreover, APAN [49] accelerates model inference through asynchronous message propagation. However,

existing T-GNNs usually rely on recursive message passing, thus suffering from high computational cost. In contrast, Zebra can improve the efficiency of T-GNNs by identifying and aggregating the most instructive temporal information.

Diffusion-based Graph Neural Networks. The idea of improving GNNs via graph diffusion is first proposed in [19], which derives a new propagation scheme based on personalized PageRank (PPR). It [19] aims to enlarge the receptive field of static GNNs without incurring the over-smoothing issue [31]. The follow-up work [8] further accelerates approximated PPR computation by parallelizing the forward push algorithm [6]. Other graph diffusion methods like heat kernel PageRank [20] and ordinary differential equation [9] are also explored in GNN computation. Moreover, the randomized algorithm AGP [43] provides a unified solution for approximately computing various proximity queries. To build more robust and power GNNs, SIGN [37] and GMLP [58] combines various diffusion methods for information propagation. In contrast to prior works, we are the first to study the diffusion process for T-GNNs. Motivated by the observation that the importance of temporal interactions generally follows exponential distributions, this paper proposes a novel metric named T-PPR. By answering top-k T-PPR queries, we can distinguish influential temporal nodes over dynamic graphs.

Personalized PageRank. PPR [32] is a critical proximity measure for static graphs. The advanced algorithms [26, 28, 46–48, 52] for fast PPR computation are the combination of basic techniques including forward push [6], reverse push [5], power iteration [32], and Monte-Carlo sampling. For instance, a line of research [26, 47, 48] combines forward push with the Monte-Carlo method to improve the computational efficiency with error guarantees. To approximate PPR computation on dynamic graphs, the previous methods [7, 29, 56, 57] incrementally update indexes (e.g., random walk segments) when the graph structure changes. As for top-k PPR query, early works [11, 12, 53] follow the branch-and-bound idea to accelerate the exact top-k computation by dynamically updating the lower and upper error bounds. Recently, approximation algorithms [40, 51] are then developed to improve the efficiency of top-k PPR query with theoretical guarantees. Despite the success of PPR, most of existing works focus on undirected and unweighted graphs and thus cannot be efficiently extended to solve our T-PPR problem. For weighted graphs, the push-based [5, 6, 47] and iteration-based [26, 52] methods would spend significant time on calculating a tiny probability mass, resulting in severe overhead. While the EdgePush algorithm [45] can address this issue, it is not applicable to dynamic graphs because this approach needs to precompute and maintain a threshold for each edge in the graph.

3 BACKGROUND

In this section, we first review continuous-time dynamic graphs (CTDGs) and temporal random walk. Based on the above two concepts, we then introduce T-GNNs, which are powerful models that learn temporal embeddings on CTDGs via recursive temporal neighborhood aggregation. Table 1 summarizes the primary notations used throughout this paper.

Table 1: Summary of primary notations.

Notation	Description
G	A continuous-time dynamic graph
$\gamma(t)$	The interaction arriving at timestamp t
t^-, t^+	The time just before and after t
v_i	A node in the dynamic graph
s_i	The state vector of node v_i
$e_{ij}(t), e(t)$	The feature of the interaction arriving at t
$(v_i, t), (i, t)$	A temporal node (node v_i at timestamp t)
$N_{i,t}$	Temporal neighbors of (i, t)
$h_{i,t}^{(l)}$	The embedding of (i, t) at layer l
$w_{i,t}(j, \tau)$	The weight of (j, τ) with respect to (i, t)
$P_{i,t}(j, \tau)$	Temporal random walk probability from (i, t) to (j, τ)
α	The termination probability in T-PPR
β	The exponential decay factor in T-PPR
$\pi_{i,t}$	The T-PPR dictionary for (i, t) with default value 0
$\tilde{\pi}_{i,t}$	The estimated T-PPR dictionary for (i, t) with default 0
$\pi_{i,t}(j, \tau)$	The T-PPR value of (j, τ) concerning (i, t)
$\tilde{\pi}_{i,t}(j, \tau)$	The estimated T-PPR value of (j, τ) concerning (i, t)

3.1 Continuous-Time Dynamic Graphs

Definition 1 (Continuous-Time Dynamic Graph (CTDG)). *A CTDG G is represented as a sequence of edge interactions $G = \{\gamma(t_1), \gamma(t_2), \dots\}$ arranged in increasing order of time. Each interaction is a quadruple $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$ representing a (directed) temporal edge, where v_i and v_j are nodes, $e_{ij}(t)$ is a vector of edge feature, and $t \in \mathbb{N}^+$ is a timestamp at which $\gamma(t)$ happens.*

In this work, we focus on CTDGs due to their generality and flexibility. For undirected CTDGs, both interactions $(v_i, v_j, e_{ij}(t), t)$ and $(v_j, v_i, e_{ij}(t), t)$ are stored. For simplicity, we refer to CTDGs as dynamic graphs when it is clear from the context. A CTDG can be a multigraph since multiple interactions may occur between two nodes at different timestamps.

3.2 Temporal Random Walk

Temporal Node. Let (v_i, t), or simply (i, t) denote a temporal node (i.e., node v_i at timestamp t). A temporal node serves as the *basic unit* of temporal random walks and T-GNN computation.

Temporal Neighbor. For any temporal node (i, t), we define its temporal neighbor set in a dynamic graph G as

$$N_{i,t} = \{(j, \tau) | \ (v_i, v_j, e_{ij}(\tau), \tau) \in G, \tau < t\}, \tag{1}$$

where $(j, \tau) \in N_{i,t}$ indicates that node v_j interacted with node v_i at a previous timestamp τ ($\tau < t$).

Temporal Random Walk. Temporal random walks [30, 50] capture the diffusion process on dynamic graphs and proceed in *inverse chronological order*. Therefore, a walk reaching (i, t), at the next step, can only move to a temporal neighbor $(j, \tau) \in N_{i,t}$ if $N_{i,t} \neq \emptyset$; otherwise, the temporal random walk has to terminate at (i, t).

3.3 Temporal Graph Neural Networks

Temporal graph neural networks (T-GNNs) are powerful models for learning representations on dynamic graphs. Existing state-of-the-art T-GNNs [22, 36, 41, 49, 54, 60] can be abstracted into a *generic* architecture, including *temporal message passing* and *state update*.

For each node v_i, T-GNNs dynamically maintain a state vector s_i, which summarizes the information of all the past interactions that involve node v_i. T-GNNs then compute temporal embeddings based on the state vectors through recursive temporal sampling and neighborhood aggregation. Specifically, given a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$, an L-layer T-GNN model works in the following two steps.

- **Temporal Message Passing.** T-GNN computes the temporal embedding $h_{i,t}^{(L)}$ of node v_i at timestamp t by *recursively aggregating its L-hop temporal neighborhood* as follows:

$$\tilde{N}_{i,\tau} = \mathsf{SAMPLE}(N_{i,\tau}), \tag{2}$$

$$h_{i,\tau}^{(l)} = \mathsf{AGGREGATE}(\{h_{j,\tau'}^{(l-1)} \parallel e_{ij}(\tau') \parallel \phi(t-\tau') | (j,\tau') \in \tilde{N}_{i,\tau}\}), \forall l = 1, \ldots, L, \tag{3}$$

$$h_{i,\tau}^{(0)} = s_i. \tag{4}$$

  To compute $h_{i,\tau}^{(l)}$ ($l \geq 1$) of (i,τ) at layer l, T-GNN first samples a set of temporal neighbors $\tilde{N}_{i,\tau}$ that have interacted with v_i before τ (Eq.2). Then, T-GNN aggregates sampled neighborhood features, which are represented by the concatenation of $(l-1)$-layer embedding $h_{j,\tau'}^{(l-1)}$, edge feature $e_{ij}(\tau')$, and time encoding $\phi(t-\tau')$ (Eq.3). Here, the function $\phi(\cdot)$ encodes the delta time between the target timestamp t and a sampled interaction $\gamma(\tau')$. Moreover, T-GNN aggregates state vectors to compute the first-layer embeddings (Eq.4). As shown in Figure 1a, the computation graphs of T-GNNs are *time-dependent*, and each target timestamp t is associated with a *unique* computation graph.
- **State Update.** After a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$ arrives, T-GNN updates state vectors of nodes v_i and v_j, respectively. For instance, the state vector s_i of node v_i is updated as

$$s_i = \mathsf{UPDATE}(s_i, s_j, e_{ij}(t), \phi(t-t')), \tag{5}$$

  where s_i and s_j on the right-hand side are the latest state vectors before the interaction $\gamma(t)$ arrives, and t' ($t' < t$) denotes the last update timestamp of node v_i.

Time Complexity. Let k be the limit on the number of temporal neighbors sampled for each temporal node in Eq.2 and L be the number of layers. Then, there are $O(k^{L-l})$ ($1 \leq l \leq L$) nodes at the l-th layer. Moreover, T-GNN needs to aggregate neighborhood features of at most k temporal neighbors to compute the representation of a temporal node. Therefore, the time complexity of an L-layer T-GNN is $O(\sum_{l=1}^{L} k \cdot k^{L-l}) = O(k^L)$.

4 FROM T-GNN TO T-PPR

As described in Section 3.3, vanilla T-GNNs take state vectors (Eq.4) and edge features (Eq.3) as inputs to compute temporal embeddings. In this section, we first analyze how much each input feature affects the model output (Section 4.1 and Section 4.2). Based on the influence analysis, we then propose T-PPR and discuss the opportunities of improving T-GNNs via T-PPR (Section 4.3).

4.1 Preliminaries

Influence Functions. Inspired by [21, 55, 59], we analyze the influence of an input feature x on model output y by measuring the *magnitude of the corresponding partial derivatives* $\partial y/\partial x$. More specifically, we define the influence score of state vectors and edge features in Def. 2 and Def. 3, respectively.

Definition 2 (Influence Score of State Vectors). *Let $h_{i,t}^{(L)}$ be the temporal embedding of (i,t) computed by an L-layer T-GNN and S be the set of state vectors involved in the computation of $h_{i,t}^{(L)}$. The influence score of any state vector $s_j \in S$ on $h_{i,t}^{(L)}$ is the norm of the expected Jacobian matrix $I((i,t), s_j) = \|E[\partial h_{i,t}^{(L)} / \partial s_j]\|$. Then, the normalized influence score of s_j can be defined as*

$$I_{i,t}(s_j) = \frac{I((i,t),s_j)}{\sum_{s_z \in S} I((i,t),s_z)}. \tag{6}$$

Definition 3 (Influence Score of Edge Features). *Let $h_{i,t}^{(L)}$ be the temporal embedding of (i,t) computed by an L-layer T-GNN and E be the set of edge features involved in the computation of $h_{i,t}^{(L)}$. The influence score of any edge feature $e_{jz}(\tau) \in E$ on $h_{i,t}^{(L)}$ is the norm of the expected Jacobian matrix $I((i,t), e_{jz}(\tau)) = \|E[\partial h_{i,t}^{(L)} / \partial e_{jz}(\tau)]\|$. Then, the normalized influence score of $e_{jz}(\tau)$ can be defined as*

$$I_{i,t}(e_{jz}(\tau)) = \frac{I((i,t),e_{jz}(\tau))}{\sum_{e_{pq}(\tau') \in E} I((i,t),e_{pq}(\tau'))}. \tag{7}$$

Generic Aggregator. The analysis of influence score depends on the aggregator used in Eq.3. Without loss of generality, we study the most fundamental and generic aggregator [36] defined as below:

$$h_{i,\tau}^{(l)} = \mathsf{ReLU}(W^{(l)} \sum_{(j,\tau') \in N_{i,\tau}} w_{i,\tau}(j,\tau')(h_{j,\tau'}^{(l-1)} \parallel e_{ij}(\tau') \parallel \phi(t-\tau'))) \tag{8}$$

where $W^{(l)}$ is a learnable weight matrix at the l-th layer, $w_{i,\tau}(j,\tau')$ is the non-negative weight of the temporal neighbor (j,τ') with respect to (i,τ), and the sum of weights $\sum_{(j,\tau') \in N_{i,\tau}} w_{i,\tau}(j,\tau') = 1$. In practice, we can set uniform neighbor weights [36] or learn the weights using attention modules [54].

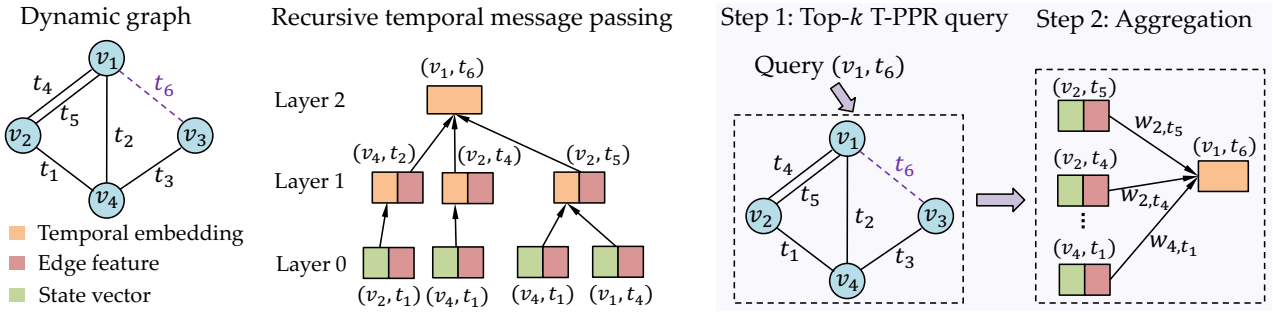
4.2 Influence Analysis

Key Intuition. We observe that the weight $w_{i,\tau}(j,\tau')$ used in the generic aggregator (Eq.8) implicitly defines the 1-step temporal random walk probability $P_{i,\tau}(j,\tau')$ from (i,τ) to (j,τ'). Then, the multiplication of multi-hop neighbor weights corresponds to the multi-hop temporal random walk probability. This finding motivates us to build a unified view of the temporal message passing scheme in T-GNN and temporal random walk on dynamic graphs. Specifically, the following theorems show that the normalized influence score of an input feature x (i.e., state vector or edge feature) on T-GNN's output embedding y is proportional to the temporal random walk probability from the temporal node associated with y to the temporal node associated with x. We defer the proofs of the theorems to our technical report [3] due to the page limit.

Theorem 1. *For an L-layer T-GNN with the generic aggregator (Eq.8), the normalized influence score $I_{i,t}(s_j)$ is proportional to the L-step temporal random walk probability from (i,t) to the node v_j.*

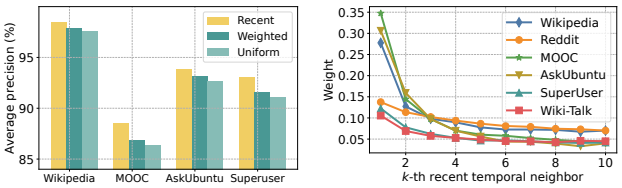
Theorem 2. *For an L-layer T-GNN with the generic aggregator (Eq.8), the normalized influence score $I_{i,t}(e_{jz}(\tau))$ is proportional to a weighted sum of l-step ($1 \leq l \leq L$) temporal random walk probability from (i,t) to (z,τ).*

Discussion. The above two theorems suggest that we can directly compute the normalized influence score of an input feature as long as there exists a method that is able to sufficiently capture



(a) Vanilla T-GNN (b) Overview of Zebra

Figure 1: (a) Example of a 2-layer vanilla T-GNN. In the dynamic graph, the dotted line with timestamp t_6 indicates a newly arrived interaction $\gamma(t_6) = (v_1, v_3, e_{13}(t_6), t_6)$. Then, the T-GNN computes the temporal embedding $h^{(2)}_{1,t_6}$ of node v_1 at timestamp t_6 through recursive message passing. Concretely, the target temporal node (v_1, t_6) has three 1-hop temporal neighbors $\{(v_4, t_2), (v_2, t_4), (v_2, t_5)\}$. The T-GNN aggregates the information (i.e., concatenation of temporal embeddings and edge features) of those temporal neighbors to compute the embedding of (v_1, t_6). Besides, the T-GNN takes state vectors and edge features as input to compute the first layer embeddings. (b) Our Zebra framework accelerates T-GNN by avoiding recursive temporal message passing. Given a target (v_1, t_6), Zebra first solves the top-k T-PPR query and returns a set of temporal nodes that are most influential to (v_1, t_6). Then, Zebra aggregates the state vectors and edge features of the returned top-k nodes. Here, the weight w used in neighborhood aggregation corresponds to the normalized T-PPR value of a temporal node.



(a) T-GNN with different samplers (b) Learned neighbor weights

Figure 2: (a) Average precision of vanilla T-GNN varying neighbor samplers on four datasets. (b) Weights of the top-10 recent neighbors learned by vanilla T-GNN on six datasets.

the diffusion process (i.e., temporal random walk probabilities) on dynamic graphs. Then, it would be possible to directly select a few prominent temporal neighbors with large influence scores to compute a temporal node's embedding. In this way, we can prune insignificant neighbors and avoid the laborious process of recursive neighborhood aggregation, thus accelerating T-GNN computation.

4.3 Improving T-GNN via T-PPR

Inspired by the above theoretical findings, we aim to estimate the temporal random walk probabilities based on neighbor weights learned by T-GNNs for computing the influence score of nodes on evolving graphs. As we cannot know the learned neighbor weights in advance, we get key insight from the behavior of vanilla T-GNN, which recursively aggregates sampled temporal neighborhood features via an attention module [42]. Figure 2a shows the impact of three neighbor samplers (i.e., recent sampling, weighted sampling, uniform sampling) on the average precision of a 2-layer vanilla T-GNN [36]. Given a target node, the weighted sampling approach [50] assigns recently interacted neighbors higher probabilities following an exponential distribution; the recent sampling approach [36] simply selects the top-k recent temporal neighbors for feature aggregation. Figure 2a suggests that the most recent temporal neighbors play a more crucial role in learning high-quality temporal embeddings since the recent sampling approach clearly outperforms the other neighbor samplers in model performance. Moreover, Figure 2b exhibits the weights of the top-10 recent temporal neighbors learned by T-GNN on six real-world graphs. Notably, the weights approximately follow exponential distributions irrespective of the input edge features and state vectors.

4.3.1 Temporal Personalized PageRank (T-PPR). Based on the above observations, we propose temporal personalized PageRank (T-PPR) to estimate the temporal random walk probabilities over dynamic graphs. Without loss of generality, we assume that there is no sink node [44] in the graph by conceptually linking sink nodes to a dummy node, which is not involved in T-PPR computation. T-PPR consists of the following two components.

- **α-Temporal Random Walk.** Denote by α $(0 < \alpha < 1)$ the termination probability of temporal random walks on dynamic graphs. Then, an α-temporal random walk starting from (i, t) reaching (j, τ), at the next step, can (i) terminate at (j, τ) with probability α, or (ii) move to a temporal neighbor (the dummy node) with probability $1 - \alpha$ if $N_{j,\tau} \neq \emptyset$ $(N_{j,\tau} = \emptyset)$.
- **β-Exponential Decay.** For any temporal node (i, t) with non-empty neighbor set, we define the transition probability from (i, t) to one of its temporal neighbor $(j, \tau) \in N_{i,t}$ as

$$P_{i,t}(j, \tau) = \frac{\beta^{|\{(j',\tau')|(j',\tau')\in N_{i,t}, \tau' \geq \tau\}|}}{\sum_{z=1}^{|N_{i,t}|} \beta^z}, \tag{9}$$

where β $(0 < \beta < 1)$ is an exponential decay factor assigning the k-th recent temporal neighbor transition probability proportional

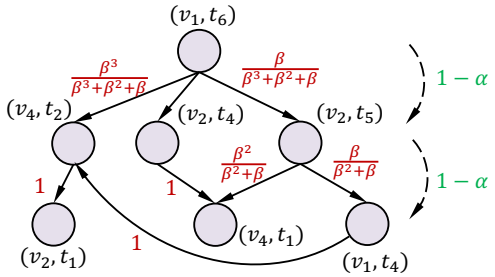


Figure 3: An illustration of T-PPR. The green notes indicate that temporal random walks starting from (v_1, t_6) move to a temporal neighbor with probability $1-\alpha$, and the red notes show the probabilities of taking different temporal interactions at the next step.

to β^k. Our intuition is that recent interactions capture the latest trend and thus convey more information in the diffusion process on dynamic graphs. Moreover, the exponential decay model has been widely adopted in temporal neighbor sampling [30, 50], stream processing [38], and social network analysis [13, 14].

Definition 4 (Temporal Personalized PageRank (T-PPR)). *Consider a dynamic graph G. Given a source (i,t) and a target (j,τ), the T-PPR value of (j,τ) with respect to (i,t), denoted by $\pi_{i,t}(j,\tau)$, is defined as the probability that an α-temporal random walk with β-exponential decay starting from (i,t) terminates at (j,τ).*

Definition 5 (Top-k T-PPR Query). *Consider a dynamic graph G. Given a temporal node (i,t), the goal of top-k T-PPR query is to return a dictionary $\pi_{i,t}$ of temporal nodes (excluding (i,t)) that have the top-k largest T-PPR values with respect to (i,t). For a top-k temporal node $(j,\tau) \in \pi_{i,t}$, $\pi_{i,t}(j,\tau)$ denotes its T-PPR value.*

Example 3. Figure 3 demonstrates how T-PPR models the diffusion process on the dynamic graph shown in Figure 1a. The source node (v_1, t_6) has three temporal neighbors, where the most recent temporal neighbor (v_2, t_5) is assigned transition probability $P_{1,t_6}(2, t_5) \propto \beta$. Hence, the T-PPR value of (v_2, t_5) with respect to (v_1, t_6) can be calculated as $\pi_{1,t_6}(2, t_5) = \beta/(\beta^3+\beta^2+\beta)(1-\alpha)\alpha$. Particularly, (v_2, t_1) is a sink node without temporal neighbors. Hence, temporal random walks reaching (v_2, t_1) terminate with probability α or move to a dummy node.

Table 2: Comparison of PPR and T-PPR.

Attribute	Graph	Basic unit	Edge weight	Topology
PPR	Static	Node	Usually unweighted	Cyclic
T-PPR	Dynamic	Node-time	Exponential	Acyclic

Comparison of PPR and T-PPR. Table 2 summarizes the key differences between T-PPR and classical personalized PageRank (PPR). In general, PPR [32] measures the proximity score between nodes on static graphs. Most of existing works [26, 47, 48, 52] on PPR assume the input graph is unweighted and cyclic. In contrast, T-PPR is a novel proximity measure that captures the diffusion process on dynamic graphs via an exponential decay model. Moreover, temporal random walks in T-PPR produce acyclic graphs because these walks proceed in inverse chronological order.

4.3.2 Improving T-GNN via T-PPR. The T-PPR value $\pi_{i,t}(j,\tau)$ defined in Def. 4 measures the probability that an α-temporal random walk with β-exponential decay starting from (i,t) terminates at (j,τ). Based on Theorem 1 and Theorem 2, $\pi_{i,t}(j,\tau)$ corresponds to the influence score of the features associated with (j,τ) on vanilla T-GNN's output embedding of (i,t). Therefore, given a target (i,t), we search and aggregate the top-k most influential temporal nodes (i.e., nodes with the top-k largest T-PPR scores), thus improving the computational efficiency of T-GNNs.

Besides its efficiency, T-PPR could also improve the predictive performance of T-GNNs in downstream tasks. Simply aggregating multi-hop temporal neighborhood features as vanilla T-GNNs may hurt model accuracy due to the over-smoothing issue [10, 24, 31]. Concretely, over-smoothing indicates that a target node's relevant local neighborhood information may be "washed out" by insignificant neighbor nodes. Fortunately, T-PPR can help overcome the over-smoothing issue by directly ruling out the irrelevant temporal nodes that have small T-PPR values.

5 THE ZEBRA FRAMEWORK

5.1 Framework Overview

In this section, we present Zebra, a framework that improves T-GNN via T-PPR. Figure 1b gives an overview of Zebra. Without loss of generality, Zebra needs to work in the following two steps.

- **Top-k T-PPR Query.** Given a newly arrived temporal node (i,t), Zebra first solves the top-k T-PPR query and returns a dictionary $\pi_{i,t}$, which records the top-k influential temporal nodes with respect to (i,t) and the corresponding T-PPR values. The technical challenge here is how to efficiently answer the top-k query and eventually achieve end-to-end speedup of T-GNN computation. To solve this problem, we propose an efficient single-scan algorithm with approximation guarantees in Section 6. Moreover, we empirically show that a small value of k can already lead to remarkable model performance in Section 7.
- **1-layer Neighborhood Aggregation.** After obtaining the top-k T-PPR dictionary $\pi_{i,t}$, Zebra then computes the temporal embedding $h_{i,t}$ for the target temporal node (i,t) as follows:

$$w_{j,\tau} = \frac{\pi_{i,t}(j,\tau)}{\sum_{(j,\tau)\in\pi_{i,t}} \pi_{i,t}(j,\tau)}, \tag{10}$$

$$h_{i,t} = \sum_{(j,\tau)\in\pi_{i,t}} w_{j,\tau} \cdot \mathsf{Transform}(s_j \,\|\, e(\tau) \,\|\, \phi(t-\tau)). \tag{11}$$

Zebra first calculates the normalized weight $w_{j,\tau}$ for each selected temporal node (j,τ) so that the total weights sum to 1 (Eq.10). Then, Zebra aggregates the features of the top-k temporal nodes with the normalized weights and generate the temporal embedding $h_{i,t}$ (Eq.11). Here, for each selected temporal node (j,τ), the Transform function encodes the corresponding state vector s_j, edge feature $e(\tau)$, and time feature $\phi(t-\tau)$ as vanilla T-GNN. In our experiments, we find the Transform function implemented as a fully connected layer yields good performance.

Time Complexity. The complexity of Zebra for computing the embedding of a single node is $O(C(k)+k)$, where $C(k)$ denotes the cost

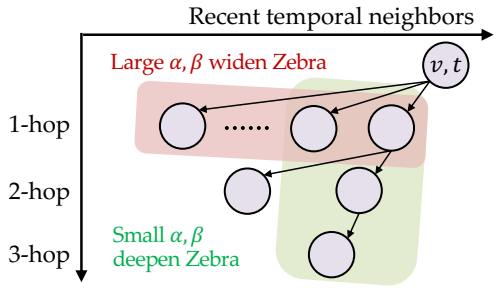


Figure 4: An illustration of the ensemble idea. The shades of red and green indicate the top-k influential temporal neighbors of (v, t) selected by two different T-PPR metrics. In theory, an ensemble of diversified T-PPR metrics can widen and deepen the receptive field of Zebra at the same time, thus boosting and stabilizing model performance.

Algorithm 1: Single-scan top-k T-PPR algorithm (SANTA)

input : $\{(\tilde{\pi}_{i,t^-}, m_{i,t^-}) | v_i \in V\}$, new interaction $(v_i, v_j, e_{ij}(t), t)$, threshold k

ouput : Approximated top-k T-PPR dictionary

$\bar{\pi}_{i,t^+} \leftarrow \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\tilde{\pi}_{i,t^-} + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}(\tilde{\pi}_{j,t^-} + \alpha I_{j,t})$

$\tilde{\pi}_{i,t^+} \leftarrow$ temporal nodes in $\bar{\pi}_{i,t^+}$ with the top-k largest estimated T-PPR values

if $v_i \neq v_j$ **then**

  $\bar{\pi}_{j,t^+} \leftarrow \frac{m_{j,t^-}\beta}{m_{j,t^-}\beta+\beta}\tilde{\pi}_{j,t^-} + \frac{\beta(1-\alpha)}{m_{j,t^-}\beta+\beta}(\tilde{\pi}_{i,t^-} + \alpha I_{i,t})$

  $\tilde{\pi}_{j,t^+} \leftarrow$ temporal nodes in $\bar{\pi}_{j,t^+}$ with the top-k largest estimated T-PPR values

  $m_{j,t^+} \leftarrow m_{j,t^-}\beta + \beta$

$m_{i,t^+} \leftarrow m_{i,t^-}\beta + \beta$

return $\tilde{\pi}_{i,t^+}, \tilde{\pi}_{j,t^+}$

of top-k T-PPR query, and $O(k)$ corresponds to the cost of neighborhood aggregation (Eq.10, Eq.11). We later develop an efficient approximation algorithm (Algorithm 1) with $C(k) = O(k \log k)$ in Section 6. In contrast, as discussed in Section 3.3, the complexity of vanilla T-GNN is $O(k^L)$. Hence, Zebra equipped with Algorithm 1 can be almost exponentially faster than vanilla T-GNNs.

5.2 Ensemble of T-PPR

The configuration of the T-PPR used in Zebra is important since the quality of generated temporal embeddings highly depends on the choice of T-PPR parameters α and β. A single T-PPR, if not well parameterized, may fail to sufficiently capture the diffusion process on evolving graphs, thus degrading model performance.

To address the above limitation and free practitioners from laborious parameter tuning, we propose to build robust T-GNNs through uniting the expressive power of diversified T-PPR metrics. As shown in Figure 4, T-PPR metrics, parameterized by different α and β values, are endowed with distinct receptive fields. In general, T-PPR configured by large α and β can widen the receptive field of Zebra, while small α and β deepen the receptive field. According to the ensemble theory [39, 61], it would be beneficial to enhance Zebra by integrating the advantages of diversified T-PPR metrics.

Based on the above discussion, Zebra constructs an ensemble of temporal embeddings obtained with n T-PPR metrics as follows:

$$h_{i,t} = \text{Combine}(h_{i,t}(\alpha_1, \beta_1) \,\|\, \dots \,\|\, h_{i,t}(\alpha_n, \beta_n)), \tag{12}$$

where $h_{i,t}(\alpha_j, \beta_j)$ $(j = 1, \dots, n)$ corresponds to the temporal embedding obtained with a single T-PPR (parameterized by α_j and β_j) as shown in Eq.11, and $h_{i,t}$ is the final generated temporal embedding of (i, t). Moreover, the Combine function in Eq.12 can be simply implemented as a fully connected layer. In practice, we can parallelize the computation of multiple T-PPR metrics to reduce the extra runtime overhead. We empirically investigate the sensitivity of T-PPR parameters and the impact of ensemble in Section 7.

6 TOP-k T-PPR QUERY

While a long line of research studies efficient PPR query processing, it falls short of answering T-PPR queries due to the *constantly changing graph structure* and the *highly imbalanced weight distribution*. To compute PPR values on dynamic graphs, the previous methods [7, 29, 57] incrementally update indexes (e.g., random walk segments) when the graph structure changes. However, most of the existing works on PPR focus on undirected and unweighted graphs. For weighted graphs, the push-based [6, 7, 29, 46, 47, 57] and iteration-based [26, 32, 52] methods would spend significant time on calculating a tiny probability mass, resulting in severe overhead. Please refer to Section 2 for more detailed discussions on the difference between PPR and T-PPR.

Fortunately, we can leverage the *acyclic property* of T-PPR to develop efficient solutions. In this section, we first derive a decomposition theorem for updating T-PPR values over dynamic graphs (Section 6.1). We then propose a *single-scan* algorithm (Section 6.2) to answer the *up-to-date* top-k T-PPR query with rigorous approximation guarantees (Section 6.3).

6.1 Decomposition Theorem

A straightforward solution to the top-k T-PPR query is power iteration [32], which recursively traverses the graph and then selects the top-k temporal nodes. However, this approach is computationally prohibitive (see detailed empirical results in Table **??**). Therefore, we develop the following decomposition theorem for T-PPR, which serves as the foundation of the single-scan algorithm that we will later discuss in Section 6.2.

Theorem 4 (Decomposition Theorem). *Given a dynamic graph $G = \{\gamma(t_1), \gamma(t_2), \dots\}$ and a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$. Denote by π_{i,t^-} the T-PPR dictionary for (i, t^-) with temporal nodes as keys and T-PPR scores as values. We have the following decomposition rule for answering the T-PPR query with respect to (i, t^+):*

$$\pi_{i,t^+} = \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\pi_{i,t^-} + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}(\pi_{j,t^-} + \alpha I_{j,t}), \tag{13}$$

where t^- (t^+) denotes the time just before (after) t, $m_{i,t^-} = \sum_{z=1}^{|N_{i,t^-}|}\beta^z$ corresponds to the denominator used in the exponential decay model, $I_{j,t}$ is a dictionary that has only one key (j, t) with value 1, and the symbol "+" indicates summing dictionary values with the same key.

Discussion. Theorem 4 suggests that we can efficiently answer T-PPR queries upon new interactions in a streaming fashion. Specifically, we initialize an empty T-PPR dictionary for each newly appeared node. Given a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$, we can answer the T-PPR query with respect to (v_i, t) by returning the dictionary π_{i,t^-}. We obtain the updated dictionary π_{i,t^+} for node v_i by merging π_{i,t^-} and π_{j,t^-}. However, it is costly to dynamically maintain for each node a full T-PPR dictionary, whose size potentially grows with the number of temporal interactions in the evolving graph. It is also unnecessary to materialize the full T-PPR dictionaries since we focus on top-k T-PPR query in this work.

6.2 Single-scan Top-k T-PPR Algorithm (SANTA)

Based on the above discussion, we develop an efficient **s**ingle-sc**an** **t**op-k T-PPR **a**lgorithm (SANTA) that greedily prunes insignificant temporal nodes, as shown in Algorithm 1. We elaborate the data structure maintained by SANTA and how it works as follows.

- **Data Structure.** Let V be a set of nodes in the dynamic graph G. For each node $v_i \in V$, SANTA maintains a tuple $(\tilde{\pi}_{i,t^-}, m_{i,t^-})$, where $\tilde{\pi}_{i,t^-}$ is the latest approximated top-k T-PPR dictionary for node v_i before t, and m_{i,t^-} corresponds to the denominator used in the exponential decay model (Eq.9).
- **Singe-edge Update.** Given a new interaction $(v_i, v_j, e_{ij}(t), t)$, SANTA first computes the temporary dictionary $\bar{\pi}_{i,t^+}$ for node v_i by merging $\tilde{\pi}_{i,t^-}$, $\tilde{\pi}_{j,t^-}$, and $I_{j,t}$ (line 1). This step follows the decomposition property of T-PPR described in Theorem 4. SANTA next selects from $\bar{\pi}_i$ the top-k temporal nodes, and obtains the latest approximated top-k T-PPR dictionary $\tilde{\pi}_{i,t^+}$ (line 2). Finally, SANTA updates the value of the denominator used in Eq.9 (line 7). If the newly arrived interaction is not a self-loop (i.e., $v_i \neq v_j$), SANTA updates the data structure for node v_j similarly (lines 3-6). Following prior works [22, 36, 54], we assume that each interaction has a unique timestamp. Thus, given a batch of interactions, SANTA processes them one-by-one.

Complexity. We next analyze the time and space complexity of SANTA. Given a new interaction, the cost of merging materialized top-k T-PPR dictionaries (line 1) is $O(k)$, and the cost of top-k selection (line 2) is $O(k \log k)$. Then, the time complexity of SANTA for answering a top-k T-PPR query is $O(k \log k)$. For each node $v_i \in V$, SANTA maintains an approximated top-k T-PPR dictionary $\tilde{\pi}_i$ and a denominator m_i. Hence, the space complexity is $O(|V| \cdot k)$.
Justification for SANTA. Note that SANTA works in a streaming fashion and only answers the up-to-date T-PPR query. That is to say, for each node v_i, SANTA dynamically maintains the approximated top-k T-PPR dictionary $\tilde{\pi}_{i,t}$ with respect to the latest timestamp t instead of supporting the query of any past state (i, t') $(t' < t)$. We clarify that the single-scan approach can well satisfy the requirements of T-GNNs' offline training and online inference paradigms. Specifically, for offline T-GNN training, one chronologically splits the dynamic graph into multiple training batches, each containing a set of edge interactions. Then, T-GNNs sequentially run on these batches and compute the temporal embedding for each temporal node appeared in the interactions. Hence, T-GNNs proceed in a streaming fashion and only query the top-k T-PPR values with respect to the current timestamp. Moreover, the online inference of T-GNNs acts in the same way. Based on the above discussions, it is evident that SANTA can seamlessly support T-GNN computation.

6.3 Theoretical Guarantees

In this section, we develop theoretical analysis of SANTA. For any top-k temporal node returned by SANTA, Theorem 6 gives a lower bound of its approximated T-PPR value. Moreover, Theorem 7 and Theorem 8 bound the estimation error introduced by SANTA and investigate how the error changes with input parameters α, β, k. Please refer to our technical report [3] for the detailed proof.

Lemma 5. *Given a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$, let $\tilde{\pi}_{i,t^+}$ be the top-k T-PPR dictionary returned by SANTA. Then,*

$$\tilde{\theta}^{(k)}_{i,t^+} \geq \max\left(\frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\tilde{\theta}^{(k)}_{i,t^-}, \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}\tilde{\theta}^{(k)}_{j,t^-}\right), \tag{14}$$

where $\tilde{\theta}^{(k)}_{i,t^+}$ is the top-k T-PPR value threshold of $\tilde{\pi}_{i,t^+}$.

Theorem 6 (Lower Bound). *Given a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$, let $\tilde{\pi}_{i,t^+}$ be the top-k T-PPR dictionary returned by SANTA. For any $(z, \tau) \in \tilde{\pi}_{i,t^+}$, let R be a set of temporal random walk paths that start from (i, t^+) and terminate at (z, τ). Then,*

$$\tilde{\pi}_{i,t^+}(z, \tau) \geq \max_{r \in R} P(r), \tag{15}$$

where $P(r)$ denotes the probability of taking the path r.

Implication of Theorem 6. For any (j, τ) selected in $\tilde{\pi}_{i,t^+}$, there could be multiple temporal random walk paths from (i, t^+) to (j, τ). SANTA may omit some paths with small probabilities, thus introducing estimation errors. However, Theorem 6 shows that the approximated T-PPR value $\tilde{\pi}_{i,t}(z, \tau)$ can be lower bounded by the probability of the most probable path of temporal random walk that starts from (i, t) and stops at (z, τ). Hence, SANTA can preserve the most important temporal information over dynamic graphs.

Theorem 7 (Additive Error Bound). *Given a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$, let $\tilde{\pi}_{i,t^+}$ be the top-k T-PPR dictionary returned by SANTA. For any temporal node (z, τ), we have*

$$\tilde{\pi}_{i,t^+}(z, \tau) \leq \pi_{i,t^+}(z, \tau) \leq \tilde{\pi}_{i,t^+}(z, \tau) + \frac{\theta^{(k)}_{max}}{\alpha(1-\beta)}, \tag{16}$$

where $\theta^{(k)}_{max}$ is the maximum possible top-k threshold for the T-PPR metric parameterized by α and β.

Implication of Theorem 7. The above analysis indicates that the additive errors of SANTA are negligible when the temporal random walk probabilities are highly skewed (i.e., T-PPR is parameterized by large α or small β) or a large k is adopted.

Theorem 8 (Relative Error Bound). *Given a new interaction $\gamma(t) = (v_i, v_j, e_{ij}(t), t)$, let $\tilde{\pi}_{i,t^+}$ be the top-k T-PPR dictionary returned by SANTA. For any temporal node $(z, \tau) \in \tilde{\pi}_{i,t^+}$, we have*

$$\pi_{i,t^+}(z, \tau) \leq \max\left(\frac{\alpha(1-\alpha)(1+\beta-\alpha)}{(1+\beta)\tilde{\theta}^{(k)}_{i,t^+}}, 1\right)\tilde{\pi}_{i,t^+}(z, \tau) \tag{17}$$

where $\tilde{\theta}^{(k)}_{i,t^+}$ $(\tilde{\theta}^{(k)}_{i,t^+} > 0)$ is the top-k T-PPR value threshold of $\tilde{\pi}_{i,t^+}$.

Implication of Theorem 8. Note that the top-k threshold $\tilde{\theta}^{(k)}_{i,t^+}$ generally increases as we decrease the value of k. Therefore, the above theorem shows that the errors of estimated T-PPR values can be marginal for the most influential temporal nodes.

7 EXPERIMENTS

7.1 Experimental Settings

Datasets. Table 3 summarizes the six real-world dynamic graphs used in our experiments. Particularly, the diversity d [25, 27] is a critical metric that can help practitioners to find the suitable parameter setting. In general, high diversity indicates that a user may interact with many different items over time, while low diversity suggests that there may be frequent interactions between a certain user-item pair. MOOC, Wikipedia, and Reddit are common benchmarks for evaluating T-GNNs [22, 36, 50, 54]. Specifically, MOOC records student-course interactions on an online course platform; Wikipedia contains one month of changes made by editors; Reddit consists of one month of subreddit posts made by users. Moreover, AskUbuntu, Superuser, and Wiki-Talk are three large dynamic graphs consisting of timestamped user-user interactions collected from the corresponding platforms. For each dynamic graph shown in Table 3, we chronologically split it into training set (70%), validation set (15%), and test set (15%) as in [36, 54]. Besides, we randomly select 10% of nodes and remove the corresponding temporal interactions from training sets in order to test model performance in the inductive setting.

Compared Models and Implementation Details. We compare our proposed Zebra with five competitive T-GNNs.

- **Zebra** identifies essential neighbors via SANTA (Algorithm 1) and computes node embeddings through 1-layer neighborhood aggregation (Eq.11). In addition, Zebra updates the state vectors of nodes upon new interactions as vanilla T-GNN [36].
- JODIE [22] uses RNNs to propagate the information of temporal interactions to update node representations.
- TGAT [54] mimics the message passing scheme of static GNNs and encodes time information through random Fourier features.
- TGN [36] dynamically maintains a state vector for each node and includes previous methods [22, 41, 54] as special cases.
- CAW [50] encodes temporal neighborhood information using anonymized random walk and attention modules.
- APAN [49] accelerates model inference by decoupling graph computation and message propagation.

We implement the Zebra framework based on PyTorch [34] and the top-k T-PPR query algorithm SANTA using Numba [23], an LLVM-based Python JIT compiler. To evaluate Zebra, we set $k = 20$ for answering the top-k T-PPR query by default. We tune the values of parameters α and β used in T-PPR over the range $[0, 0.7]$ and $[0.2, 1]$, respectively. Besides, we adapt the official implementation of each compared baseline to our evaluation settings. For APAN, TGAT, and TGN, we set model depth $L = 2$ and sample size limit $k = 10$ (Eq.2) since this is a common and effective parameter setting. Besides, we sample 64 length-2 anonymized random walks for CAW as it is the default setting in the official implementation. We note that there are bugs in the officially released code of APAN. Consequently, it will leverage future information to make predictions, which is known as information leakage [36]. We have fixed the bugs in our empirical evaluation for a fair comparison.

Evaluation Metrics. Following prior works [36, 49, 50], we evaluate model performance in average precision (AP) on the test set.

Table 3: Statistics of the dynamic graphs used in our experiments. $|V|$ and $|E|$ indicate the number of nodes and temporal interactions. In addition, d_v and d_e denote the dimensionality of node and edge features, respectively. d is a metric that measures the diversity of interaction patterns in a dynamic graph. Specifically, given a temporal node (v, t) with non-empty temporal neighbor set $N_{v,t}$ and let $M_{v,t} = \{u|(u, \tau) \in N_{v,t}\}$ be the set of unique nodes in $N_{v,t}$, then the diversity [25, 27] of (v, t) is defined as $d_{v,t} = |N_{v,t}|/|M_{v,t}|$. The diversity of a dynamic graph is $d = \sum_{u \in U} d_{u,\infty}/|U|$, where U is the set of user nodes.

Dataset	$\|V\|$	$\|E\|$	d_v	d_e	Timespan	d
MOOC [22]	7,144	411,749	172	4	30 days	1.86
Wikipedia [22]	9,227	157,474	172	172	30 days	1.66
Reddit [22]	10,984	672,447	172	172	30 days	6.49
AskUbuntu [1]	159,316	964,437	172	0	2613 days	1.46
SuperUser [2]	194,085	1,443,339	172	0	2773 days	1.41
Wiki-Talk [4]	1,140,149	7,833,140	172	0	2320 days	2.02

Specifically, we consider two kinds of learning tasks, i.e., transductive learning and inductive learning. The transductive task tests model performance on nodes that have been observed during training, while the inductive task examines the learning ability of models on unseen nodes. As for efficiency, we evaluate the per-epoch training time (epoch time) and the total training time till convergence. Moreover, given a temporal node (i, t), we evaluate the relative error of estimated top-k T-PPR values $\tilde{\pi}_{i,t}$ as:

$$\frac{\mathsf{sum}(\pi_{i,t}) - \mathsf{sum}(\tilde{\pi}_{i,t})}{\mathsf{sum}(\pi_{i,t})}, \tag{18}$$

where $\mathsf{sum}(\pi_{i,t})$ ($|\pi_{i,t}| \neq 0$) and $\mathsf{sum}(\tilde{\pi}_{i,t})$ denote the sum of exact and approximated top-k T-PPR values, respectively.

Training Configurations. We train all the models following the common configurations [36, 50, 54]. Since the dynamic graphs in Table 3 mainly contain temporal interactions among nodes, we use link prediction as the downstream task and randomly generate fake links as negative training samples. T-GNNs are typically trained and evaluated with batches of interactions. We set the batch size to be 200 for model training, validation, and testing. Moreover, we set the maximum number of training epochs to be 50 with early stopping if a model's validation performance does not increase for five consecutive epochs. We tune the learning rate for all the compared models over the parameter space $\{1, 0.1, 0.01, 0.001\} \times 10^{-4}$. The experiments run on a server with 8 GeForce RTX 2080 Ti GPUs, 72 Intel(R) Xeon(R) 2.60 GHz CPUs, and 256 GB of memory. The reported results are averaged over five runs.

7.2 Key Results

Table 4 and Table 5 examine model performance in link prediction on six real-world dynamic graphs. In the following, we compare Zebra, implemented as an ensemble of two top-20 T-PPR metrics, with the baselines in efficiency and effectiveness.

Time Efficiency of Zebra. As shown in Table 4 and Table 5, Zebra is *one or two orders of magnitude faster* than the compared baselines in total and per-epoch training time. Zebra is computationally simple as it avoids the tedious process of recursive temporal message

Table 4: Comparison of T-GNNs on small dynamic graphs. Zebra is implemented as an ensemble of two top-20 T-PPR metrics. We report model performance in transductive average precision (%), inductive average precision (%), and total training time (s). In addition, we report the number of epochs required for model convergence in parentheses. The best and second-best results in each metric are marked in bold and underlined, respectively. "$*$" indicates that the performance improvement of Zebra over the best-performing baseline is statistically significant with the significance level set to be 0.05.

Model	Wikipedia			Reddit			MOOC		
	Trans AP	Induct AP	Training (#)	Trans AP	Induct AP	Training (#)	Trans AP	Induct AP	Training (#)
JODIE	95.16 ± 0.4	93.13 ± 0.5	2356.4 (18)	95.83 ± 0.3	93.20 ± 0.4	9243.64 (14)	83.26 ± 0.5	81.77 ± 0.4	5160.30 (15)
TGAT	94.26 ± 0.1	92.88 ± 0.3	2881.4 (29)	97.80 ± 0.2	96.08 ± 0.3	11933.8 (21)	70.22 ± 0.4	70.83 ± 0.5	7838.5 (25)
TGN	98.58 ± 0.1	98.05 ± 0.1	2182.7 (26)	98.66 ± 0.1	97.55 ± 0.1	11528.9 (26)	88.88 ± 1.7	88.17 ± 2.1	5086.83 (21)
APAN	96.41 ± 0.5	96.06 ± 0.4	1605.0 (21)	98.50 ± 0.2	97.62 ± 0.7	16431.8 (18)	87.02 ± 0.3	86.74 ± 0.5	3374.1 (14)
CAW	98.18 ± 0.1	98.24 ± 0.1	10175.5 (13)	98.54 ± 0.1	97.97 ± 0.1	75585.1 (16)	80.60 ± 0.4	80.18 ± 0.4	34063.9 (14)
Zebra	**98.67 ± 0.1**	**98.59 ± 0.1***	**302.6 (34)**	**98.76 ± 0.1***	**98.28 ± 0.1***	**1342.0 (31)**	**92.45 ± 0.2***	**89.56 ± 0.3**	**619.5 (25)**

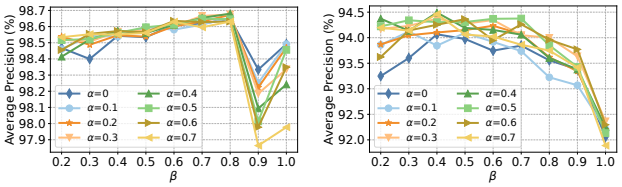
Table 5: Comparison of T-GNNs on large dynamic graphs. Zebra is implemented as an ensemble of two top-20 T-PPR metrics. The evaluation metrics are the same as those in Table 4. Particularly, "OOM" denotes out-of-memory error, and "TLE" denotes time limit exceed such that we cannot finish one epoch of model training in 12 hours. "$*$" indicates that the performance improvement of Zebra over the best-performing baseline is statistically significant with the significance level set to be 0.05.

Model	AskUbuntu			SuperUser			Wiki-Talk		
	Trans AP	Induct AP	Training (#)	Trans AP	Induct AP	Training (#)	Trans AP	Induct AP	Training (#)
JODIE	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
TGAT	87.57 ± 0.3	84.21 ± 0.4	11728.5 (21)	86.40 ± 0.5	83.12 ± 0.5	17129.3 (19)	91.72 ± 0.2	85.38 ± 0.3	189420 (34)
TGN	**94.51 ± 0.2**	92.73 ± 0.2	36202.6 (20)	93.18 ± 0.3	91.76 ± 0.2	81747.6 (24)	OOM	OOM	OOM
APAN	89.17 ± 0.1	88.43 ± 0.1	21606.1 (14)	87.07 ± 0.3	85.50 ± 0.5	48724.2 (19)	TLE	TLE	TLE
CAW	90.87 ± 0.2	90.63 ± 0.2	61903.7 (14)	88.92 ± 0.2	88.32 ± 0.1	111744.8 (15)	TLE	TLE	TLE
Zebra	94.47 ± 0.1	**97.91 ± 0.1***	**1362.2 (24)**	**93.21 ± 0.3**	**97.93 ± 0.1***	**2095.1 (23)**	**95.45 ± 0.1***	**97.96 ± 0.1***	**9909.6 (16)**

passing by efficiently solving the top-k T-PPR query. In contrast, the simplest baseline JODIE, which does not require neighborhood aggregation, is not scalable. This is because JODIE has to be trained in small batches to ensure statistical performance [22]. TGAT and TGN follow the generic T-GNN architecture described in Section 3.3, thus suffering from high computational cost. Particularly, TGN is especially slow in updating the state vectors of nodes (Eq.5) on large dynamic graphs due to poor locality of GPU memory access. Zebra resolves this problem via a lazy update strategy, i.e., only updating the state vectors of nodes that will participate in neighborhood aggregation, thus eliminating redundant computation and reducing GPU memory access. To accelerate model inference, APAN improves upon TGN by decoupling neighborhood aggregation from temporal message passing. However, APAN is extremely sensitive to the amount of available CPU and memory resources, thus suffering from inefficiency in our evaluation setting. Moreover, CAW is the slowest baseline because it adopts eight cumbersome attention modules to compute node representations. As shown in Table 5, CAW cannot even finish one epoch of model training in 12 hours on Wiki-Talk. In summary, the above experiments suggest that selecting influential temporal nodes via T-PPR can indeed improve the efficiency of T-GNNs.

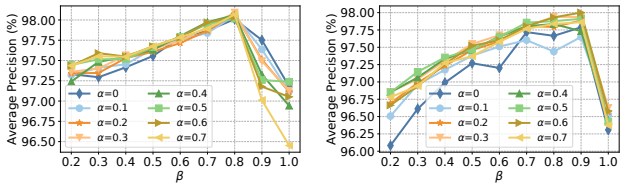
Memory Efficiency of Zebra. Table 5 shows that Zebra significantly reduces the memory consumption of T-GNNs. Specifically, JODIE and TGN suffer from excessive memory consumption on large dynamic graphs, while Zebra can efficiently compute temporal embeddings without incurring out-of-memory error. On the one hand, although SANTA requires extra memory for dynamically maintaining the top-k influential neighbors for each node over evolving graphs, we argue that the extra memory overhead can be negligible. This is because state vectors, edge features, model parameters, and gradients consume most of the memory. In practice, the dimensionality of state vectors and edge features (e.g., 172) is much larger than the value of k (e.g., 20) used for T-PPR computation. On the other hand, Zebra reduces the computational cost of T-GNNs and thus *save the GPU memory required to perform the corresponding computations*. For instance, the peak GPU memory consumption of TGN is 3GB more than that of Zebra on Wiki-Talk. Overall, the experimental results suggest that Zebra is memory efficient and scalable to large dynamic graphs (e.g., Wiki-Talk).

Effectiveness of Zebra. Though Zebra aims to make a trade-off between effectiveness and efficiency, Table 4 and Table 5 show that Zebra outperforms the state-of-the-art baselines in both transductive and inductive average precision. The same conclusion can be drawn for other evaluation metrics, including accuracy and AUC. We omit the results due to the page limit. As have discussed in Section 4.3, Zebra is more effective for two reasons. First, our proposed T-PPR is capable of capturing the relative influence of multi-hop



(a) Reddit (b) AskUbuntu

Figure 5: Transductive performance of a single top-20 T-PPR varying parameters α and β.



(a) Reddit (b) AskUbuntu

Figure 6: Inductive performance of a single top-20 T-PPR varying parameters α and β.

temporal neighbors over dynamic graphs. Second, simply aggregating the top-k influential neighbor nodes found by T-PPR can eliminate the noises introduced by insignificant nodes, thus resolving the over-smoothing issue. In contrast, vanilla T-GNNs have to recursively aggregate multi-hop neighborhood and then distinguish informative features in a data-driven manner.

Moreover, note that the advantage of Zebra is more evident and tends to be statistically significant in the inductive setting, which tests model performance on nodes that are not observed during model training. Specifically, Zebra outperforms the best baseline TGN by a large margin (i.e., 5%-6%) on AskUbuntu and SuperUser. This is because newly appeared nodes in the dynamic graph are sparsely connected. The baselines, restricted by the limited receptive field (i.e., 2-hop neighborhood), lack enough information to learn a cold node's embedding. In contrast, T-PPR enlarges the receptive field of Zebra regardless of the number of hops. We observed that a top-20 T-PPR query could find influential nodes that are 4-hops away on AskUbuntu. Hence, Zebra can capture more instructive information for cold nodes and thus generate better node representations.

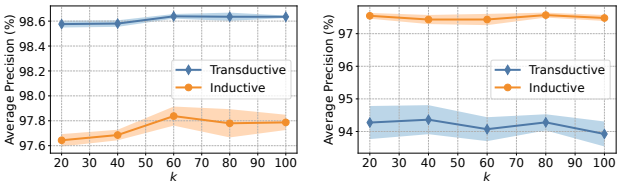
7.3 Analysis of Zebra Parameters

In this section, we conduct experiments on Wikipedia, Reddit, MOOC, and AskUbuntu to investigate the impact of Zebra parameters and provide guidance on the choice of each parameter. Due to the page limit, we report some representative results that are useful for discussions. Please see our technical report [3] for complete empirical verification and detailed discussions. In short, the guiding settings for parameter β and neighborhood size k depend on the connectivity and interaction pattern of a dynamic graph, and the choice of parameter α and ensemble size n is less sensitive to the characteristics of a specific dynamic graph.

Impact of Parameter β. A smaller β value makes T-PPR perceive a smaller neighborhood, while a larger β value makes T-PPR perceive a larger neighborhood. Figure 5 investigates the effect of parameter β on Zebra's transductive average precision. It requires a small β to achieve the best transductive performance on AskUbuntu. As shown in Table 3, AskUbuntu has low-diversity and easy-to-predict interaction patterns. In other words, a target node in this graph may repeatedly interact with very few neighbor nodes over a period of time. Hence, a small β value, which prioritizes the most recently interacted temporal neighbors only, can already lead to remarkable transductive performance. Increasing the β value may incorporate insignificant and noisy neighbors, thus deteriorating transductive performance for AskUbuntu. In contrast, a large β value is required to achieve the best transductive performance on Reddit, which has diverse interaction patterns. Given a node in Reddit, its recently interacted neighbors tend to be diverse, and there are few repeated interaction patterns. Thus, a large β value, which captures a larger neighborhood and long-term interactions, is more beneficial to learning temporal embeddings on such diverse graphs. Figure 6 further investigates the effect of parameter β on Zebra's inductive average precision. Both Reddit and AskUbuntu require a large β value in order to obtain the best inductive performance. This is because it generally takes a sufficiently large neighborhood to learn a newly appeared node's temporal representation, and a large β value enables Zebra to focus on a large neighborhood. Based on the above discussions, we suggest parameterizing a single T-PPR with a small β value (e.g., 0.5) to learn embeddings for transductive nodes with low-diversity interaction patterns. Otherwise, a large β value (e.g., 0.9) is preferred.

Impact of Parameter α. A large α value favors 1-hop temporal neighbors, while a small α value can perceive neighbors from multiple hops away. Compared with parameter β, Zebra is less sensitive to the choice of parameter α. Figure 5 and Figure 6 show that the best setting for parameter α generally lies in the range of $[0.3, 0.5]$. On the one hand, a small value of $\alpha = 0$ cannot achieve competitive performance on AskUbuntu. As suggested by Table 3, this graph has low-diversity and repeated interaction patterns. Hence, 1-hop temporal neighbors play a more crucial role in learning high-quality temporal embeddings, and a relatively large α value (e.g., 0.5) is required. On the other hand, an extremely large α value (e.g., 0.7) fails to capture the long-term temporal information from multiple hops away and could slightly hurt model performance. Based on the above discussions, we suggest setting the parameter α in the range of $[0.3, 0.5]$ in practice.

Impact of Neighborhood Size k. Figure 7 demonstrates the average precision of Zebra varying the neighborhood size k. For Reddit, there is a slight performance drop in the inductive setting when increasing k from 60 to 100. This observation indicates that simply aggregating more neighbors for computing temporal embeddings may hurt model accuracy because some insignificant and irrelevant neighbors could introduce harmful noises. In general, Zebra can achieve superior performance in the transductive setting with a small k (e.g., 20), while it may require a larger k (e.g., 60) to achieve the best inductive learning ability. This is because inductive tasks



(a) Reddit **(b) AskUbuntu**

Figure 7: Performance of a single T-PPR, parameterized by $\alpha = 0.3$ and $\beta = 0.5$, varying neighborhood size k.

test model performance on nodes that are not observed during model training. Given a newly appeared node, it generally takes a sufficiently large neighborhood to learn its temporal representation. On the other hand, the state vector of a transductive node may have already captured enough information about past temporal interactions. Hence, a small k can work well in practice. In summary, we suggest setting a small k (e.g., 20) for transductive nodes while a large k (e.g., 40) for inductive nodes. To achieve robust performance, we recommend implementing Zebra with $k = 20$ over an ensemble of two diversified T-PPR metrics.

Table 6: Performance of Zebra with $k = 20$ varying the ensemble size n. We report transductive AP (%), inductive AP (%), and per-epoch training time (s) of models.

Ensemble size	Wikipedia			AskUbuntu		
	Trans	Induct	Epoch	Trans	Induct	Epoch
1	98.59	98.41	5.72	94.37	97.75	35.34
2	98.67	98.59	8.91	94.47	98.00	56.76
3	98.64	98.61	11.93	94.21	97.76	81.71
4	98.65	98.58	15.82	94.22	97.65	102.59

Impact of Ensemble Size n. Table 6 investigates the impact of ensemble size (i.e., the number of T-PPR metrics used in Eq.12) on Zebra's performance. Clearly, an ensemble of two T-PPR metrics outperforms a single T-PPR in average precision. This observation is in line with our expectations since an ensemble of two diversified T-PPR metrics can enlarge the receptive field of Zebra, thus improving models' generalization ability. Besides, increasing the ensemble size may not improve Zebra's performance [61] for two reasons. First, increasing the ensemble size could introduce harmful noises and irrelevant temporal nodes. The noises lead to the over-smoothing issue [10] and prevent the model from focusing on the most informative temporal information. Second, we train an MLP [35] model to combine temporal embeddings obtained with various T-PPR metrics. A larger ensemble size requires a larger MLP model that has more trainable parameters. It could be difficult to effectively train an MLP model to extract useful representations from high-dimensional combined features. As for time efficiency, although increasing the ensemble size n increases the per-epoch training time, Zebra is still significantly faster than the baselines shown in Table 4 and Table 5. In summary, Zebra powered by two suitable T-PPR metrics is efficient and effective. Therefore, we recommend setting the ensemble size $n = 2$ in practice.

Discussion. Another major advantage of the ensemble scheme is that it enhances the robustness of Zebra and frees practitioners from tedious parameter tuning. As shown in Figure 2b, the weights of the top-10 recent temporal neighbors, learned by vanilla T-GNN, approximately follow various exponential distributions. In other words, the best fitting diffusion process for different dynamic graphs is different. Therefore, the key problem is how to configure the T-PPR used in Zebra to ensure model effectiveness. Fortunately, Table 4, Table 5, and Table 6 show that an ensemble of two diversified T-PPR metrics (e.g., parameterized by $\beta = 0.5$ and $\beta = 0.95$) demonstrates robust and remarkable performance on all the evaluated six real-world dynamic graphs. This is because, as illustrated in Figure 4, an ensemble of a large-β T-PPR and a small-β T-PPR can widen and deepen the receptive field of Zebra at the same time, thus improving models' robustness.

8 CONCLUSION

In this paper, we present a scalable framework Zebra that improves T-GNNs through solving the top-k T-PPR query. We theoretically prove that the influence of input node x on T-GNN's output embedding of node y is proportional to the probability that a temporal random walk starting from node y terminates at node x. Therefore, we have formally defined T-PPR, a novel metric that can effectively estimate the influence of temporal neighborhood features via an exponential decay model. We further propose an efficient single-scan algorithm to answer the top-k T-PPR query with theoretical guarantees. Zebra aggregates the most influential neighbors returned by the top-k T-PPR query, thus greatly improving the computational efficiency. Substantial experiments validate that Zebra can be up to two orders of magnitude faster than the state-of-the-art baselines while attaining better performance on various dynamic graphs. This work advances two data management research directions. We expand the scope of query processing research by proposing a novel proximity metric T-PPR, which can capture the diffusion process on dynamic graphs. Besides, the data management community is interested in developing metric-based embedding learning methods. Our work follows this routine and advances the current research to scale temporal embedding learning on evolving graphs.

ACKNOWLEDGMENTS

Yanyan Shen is supported by the National Key Research and Development Program of China (2022YFE0200500), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), and SJTU Global Strategic Partnership Fund (2021 SJTU-HKUST). Lei Chen is partially supported by National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants. Yanyan Shen is the corresponding author.

REFERENCES

[1] [2023]. AskUbuntu. http://snap.stanford.edu/data/sx-askubuntu.html.
[2] [2023]. SuperUser. http://snap.stanford.edu/data/sx-superuser.html.
[3] [2023]. The technical report. https://github.com/LuckyLYM/Zebra/blob/main/technical_report.pdf.
[4] [2023]. Wiki-talk. http://snap.stanford.edu/data/wiki-talk-temporal.html.
[5] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2008. Local Computation of PageRank Contributions. *Internet Mathematics* 5, 1 (2008), 23–45.
[6] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *FOCS*. IEEE Computer Society, 475–486.
[7] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *PVLDB* 4, 3 (2010), 173–184.
[8] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *KDD*. ACM, 2464–2473.
[9] Ben Chamberlain, James Rowbottom, Maria I. Gorinova, Michael M. Bronstein, Stefan Webb, and Emanuele Rossi. 2021. GRAND: Graph Neural Diffusion. In *ICML (Proceedings of Machine Learning Research)*, Vol. 139. PMLR, 1407–1418.
[10] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI*. AAAI Press, 3438–3445.
[11] Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka, and Masaru Kitsuregawa. 2012. Fast and Exact Top-k Search for Random Walk with Restart. *PVLDB* 5, 5 (2012), 442–453.
[12] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. 2013. Efficient ad-hoc search for personalized PageRank. In *SIGMOD*. ACM, 445–456.
[13] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. 2011. Uncovering the Temporal Dynamics of Diffusion Networks. In *ICML*. Omnipress, 561–568.
[14] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. 2010. Inferring networks of diffusion and influence. In *KDD*. ACM, 1019–1028.
[15] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge Based System* 187 (2020).
[16] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna R. Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational Graph Recurrent Neural Networks. In *NeurIPS*. 10700–10710.
[17] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.
[18] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *CoRR* abs/1611.07308 (2016).
[19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*. OpenReview.net.
[20] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *NeurIPS*. 13333–13345.
[21] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *ICML (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, 1885–1894.
[22] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *SIGKDD*. ACM, 1269–1278.
[23] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: a LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM 2015, Austin, Texas, USA, November 15, 2015*, Hal Finkel (Ed.). ACM, 7:1–7:6.
[24] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs Go As Deep As CNNs?. In *ICCV*. IEEE, 9266–9275.
[25] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. [n.d.]. A Diversity-Promoting Objective Function for Neural Conversation Models. In *HLT-NAACL*. 110–119.
[26] Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. 2020. Index-Free Approach with Theoretical Guarantee for Efficient Random Walk with Restart Query. In *ICDE*. IEEE, 913–924.
[27] Siyang Liu, Sahand Sabour, Yinhe Zheng, Pei Ke, Xiaoyan Zhu, and Minlie Huang. [n.d.]. Rethinking and Refining the Distinct Metric. In *ACL*. 762–770.
[28] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM*. ACM, 163–172.
[29] Dingheng Mo and Siqiang Luo. 2021. Agenda: Robust Personalized PageRanks in Evolving Graphs. In *CIKM*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 1315–1324.
[30] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-Time Dynamic Network Embeddings. In *WWW*. ACM, 969–976.
[31] Kenta Oono and Taiji Suzuki. 2020. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *ICLR*. OpenReview.net.
[32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Stanford InfoLab.
[33] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI*. 5363–5370.
[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*. 8024–8035.
[35] Allan Pinkus. 1999. Approximation theory of the MLP model in neural networks. *Acta numerica* 8 (1999), 143–195.
[36] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *CoRR* abs/2006.10637 (2020).
[37] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael M. Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. *CoRR* abs/2004.11198 (2020). arXiv:2004.11198
[38] Polina Rozenshtein and Aristides Gionis. 2016. Temporal PageRank. In *ECML (Lecture Notes in Computer Science)*, Vol. 9852. Springer, 674–689.
[39] Omer Sagi and Lior Rokach. 2018. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 4 (2018).
[40] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. 2019. Real-time Top-k Personalized PageRank over Large Graphs on GPUs. *PVLDB* 13, 1 (2019), 15–28.
[41] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.
[42] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[43] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2021. Approximate Graph Propagation. In *KDD*. ACM, 1686–1696.
[44] Hongwei Wang and Jure Leskovec. 2020. Unifying Graph Convolutional Neural Networks and Label Propagation. *CoRR* abs/2002.06755 (2020).
[45] Hanzhi Wang, Zhewei Wei, Junhao Gan, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2022. Edge-based Local Push for Personalized PageRank. *CoRR* abs/2203.07937 (2022).
[46] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective Indexing for Approximate Personalized PageRank. *PVLDB* 10, 3 (2016), 205–216.
[47] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries. *ACM Transactions on Database Systems* 44, 4 (2019), 18:1–18:37.
[48] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*. ACM, 505–514.
[49] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, and Zhenyu Guo. 2021. APAN: Asynchronous Propagation Attention Network for Real-time Temporal Graph Embedding. In *SIGMOD*. ACM, 2628–2638.
[50] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *ICLR*.
[51] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 441–456.
[52] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *SIGMOD*. ACM, 1996–2008.
[53] Yubao Wu, Ruoming Jin, and Xiang Zhang. 2014. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *SIGMOD*. ACM, 1139–1150.
[54] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. In *ICLR*.
[55] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML (Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 5449–5458.
[56] Minji Yoon, Woojeong Jin, and U Kang. 2018. Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees. In *WWW*. ACM, 409–418.
[57] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate Personalized PageRank on Dynamic Graphs. In *KDD*. ACM, 1315–1324.

[58] Wentao Zhang, Yu Shen, Zheyu Lin, Yang Li, Xiaosen Li, Wen Ouyang, Yangyu Tao, Zhi Yang, and Bin Cui. 2021. GMLP: Building Scalable and Flexible Graph Neural Networks with Feature-Message Passing. *CoRR* abs/2104.09880 (2021).
[59] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. 2021. Grain: Improving Data Efficiency of Graph Neural Networks via Diversified Influence Maximization. *PVLDB* 14, 11 (2021), 2473–2482.
[60] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. TGL: A General Framework for Temporal GNN Training on Billion-Scale Graphs. *PVLDB* 15, 8 (apr 2022), 1572–1580.
[61] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms*. Vol. 14.