Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank

Yiming Li[1], Yanyan Shen[2], Lei Chen[1], Mingxuan Yuan[3]
[1]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[2]Shanghai Jiao Tong University, Shanghai, China
[3]Huawei Noah's Ark Lab, Hong Kong SAR, China
[1]{yliix, leichen}@cse.ust.hk, [2]shenyy@sjtu.edu.cn, [3]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, 14(1): XXX-XXX, 2023.

PVLDB Artifact Availability:
The source code is available at https://github.com/LuckyLYM/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) [23, 35, 48, 49, 53] 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 [35, 49] shows that T-GNNs can significantly outperform the static GNNs [17, 19, 41] 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 2a, 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 [49] fails to finish one epoch of model training on the Wiki-Talk (7.8M edges and 1.14M nodes) dataset [3] with an RTX 2080 Ti GPU in 12 hours; TGN [35] suffers from slow inference on large dynamic graphs due to the excessive memory consumption caused by recursive 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 [20] 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 [7, 20, 21, 42] 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 [7, 20] can be orders of magnitude faster than the vanilla GNNs [17, 41] 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 [9, 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 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 [22, 54, 58]. 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

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

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 the exponential distribution (see details in Section 3.3). 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 present Zebra, a scalable framework that improves T-GNN via T-PPR. 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. However, there remain two challenges in making Zebra efficient and effective. **First**, we need an efficient solution to the top-k T-PPR query. Otherwise, the cost saved from avoiding recursive neighborhood aggregation could be over-consumed by the query algorithm itself. Although many advanced techniques [26, 28, 45–47, 51] have been developed for PPR, they cannot be adapted to efficiently answer the top-k T-PPR query. Existing PPR solutions usually assume that the input static graph is unweighted and cyclic, while T-PPR handles highly skewed and acyclic graphs. 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. **Second**, 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. Therefore, we propose to build robust T-GNN by constructing an ensemble of temporal embeddings obtained with various 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 in both link prediction and dynamic node classification tasks.

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

2 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-PPR and T-GNN. Specifically, T-PPR is a novel proximity metric that can identify influential nodes on CTDGs based on temporal random walk probabilities; T-GNNs are powerful models that learn temporal embeddings on CTDGs via recursive temporal neighborhood aggregation. Table 1 summarizes the primary notations used throughout this paper.

2.1 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.

2.2 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

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

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)

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, 49] 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).

2.3 Temporal Personalized PageRank

In this subsection, we formally define temporal personalized PageRank (T-PPR), a novel proximity measure on dynamic graphs. Without loss of generality, we assume that there is no sink node [43] in the graph by conceptually linking sink nodes to a dummy node, which is not involved in T-PPR computation.

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

$$P_{i,t}(j, \tau) = \frac{\beta^{|\{(j', \tau') | (j', \tau') \in N_{i,t}, \tau' \geq \tau\}|}}{\sum_{z=1}^{|N_{i,t}|} \beta^z}, \tag{2}$$

where β ($0 < \beta < 1$) is an exponential decay factor assigning the k-th recent temporal neighbor transition probability proportional to β^k. Our key 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, 49], stream processing [37], and social network analysis [13, 14].

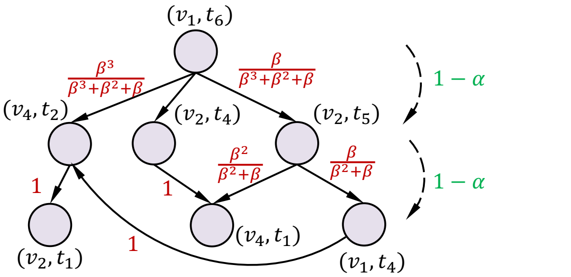

(v_1, t_6)
(v_4, t_2)
(v_2, t_4)
(v_2, t_5)
(v_2, t_1)
(v_4, t_1)
(v_1, t_4)
$\frac{\beta^3}{\beta^3+\beta^2+\beta}$
$\frac{\beta}{\beta^3+\beta^2+\beta}$
$\frac{\beta^2}{\beta^2+\beta}$
$\frac{\beta}{\beta^2+\beta}$
1
1
1
$1 - \alpha$
$1 - \alpha$

Figure 1: 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.

Definition 2 (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 3 (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 1. Figure 1 demonstrates how T-PPR models the diffusion process on the dynamic graph shown in Figure 2a. 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.

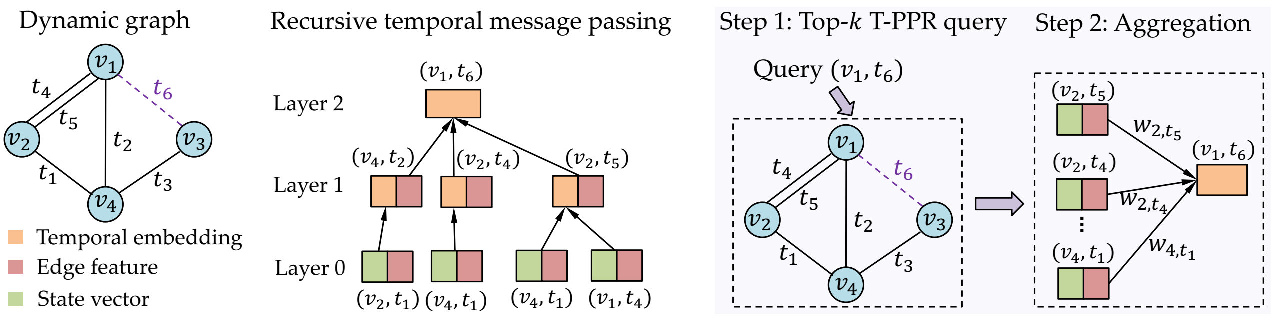
Table 2: Comparison of PPR and T-PPR.

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, 46, 47, 51] 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.

2.4 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 [23, 35, 40, 48, 53, 59] 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



(a) Vanilla T-GNN

(b) Overview of Zebra

Figure 2: (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.

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 $h^{(L)}_{i,t}$ of node v_i at timestamp t by *recursively aggregating its L-hop temporal neighborhood* as follows:

$$\tilde{N}_{i,\tau} = \mathsf{SAMPLE}(N_{i,\tau}), \tag{3}$$

$$h^{(l)}_{i,\tau} = \mathsf{AGGREGATE}(\{h^{(l-1)}_{j,\tau'} \,\|\, e_{ij}(\tau') \,\|\, \phi(t-\tau') | (j,\tau') \in \tilde{N}_{i,\tau}\}), \forall l = 1, \dots, L, \tag{4}$$

$$h^{(0)}_{i,\tau} = s_i. \tag{5}$$

  To compute $h^{(l)}_{i,\tau}$ ($l \geq 1$) 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.3). Then, T-GNN aggregates sampled neighborhood features, which are represented by the concatenation of $(l-1)$-layer embedding $h^{(l-1)}_{j,\tau'}$, edge feature $e_{ij}(\tau')$, and time encoding $\phi(t-\tau')$ (Eq.4). 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.5). As shown in Figure 2a, 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

$$s_i = \mathsf{UPDATE}(s_i, s_j, e_{ij}(t), \phi(t-t')), \tag{6}$$

  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.3 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 $O(\sum_{l=1}^{L} k \cdot k^{L-l}) = O(k^L)$.

3 FROM T-GNN TO T-PPR

As described in Section 2.4, vanilla T-GNNs take state vectors (Eq.5) and edge features (Eq.4) as inputs to compute temporal embeddings. In this section, we first analyze how much each input feature affects the model output (Section 3.1 and Section 3.2). Based on the influence analysis, we then discuss the opportunities of improving T-GNNs via T-PPR on dynamic graphs (Section 3.3).

3.1 Preliminaries

Influence Functions. Inspired by [22, 54, 58], 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. 4 and Def. 5, respectively.

Definition 4 (Influence Score of State Vectors). *Let $h^{(L)}_{i,t}$ 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 $h^{(L)}_{i,t}$. The influence score of any state vector $s_j \in S$ on $h^{(L)}_{i,t}$ is the norm of the expected Jacobian matrix $I((i,t), s_j) = \|E[\partial h^{(L)}_{i,t} / \partial s_j]\|$. Then, the normalized influence score of s_j can be defined as*

$$I_{i,t}(s_j) = \frac{I((i,t),s_j)}{\sum_{s_z \in S} I((i,t),s_z)}. \tag{7}$$

Definition 5 (Influence Score of Edge Features). *Let $h^{(L)}_{i,t}$ 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 $h^{(L)}_{i,t}$. The influence score of any edge feature $e_{jz}(\tau) \in E$ on $h^{(L)}_{i,t}$ is the norm of the expected Jacobian matrix $I((i,t), e_{jz}(\tau)) = \|E[\partial h^{(L)}_{i,t} / \partial e_{jz}(\tau)]\|$.*

Then, the normalized influence score of $e_{jz}(\tau)$ can be defined as

$$I_{i,t}(e_{jz}(\tau)) = \frac{I((i,t), e_{jz}(\tau))}{\sum_{e_{pq}(\tau') \in E} I((i,t), e_{pq}(\tau'))}. \tag{8}$$

Generic Aggregator. The analysis of influence score depends on the aggregator used in Eq.4. Without loss of generality, we study the most fundamental and generic aggregator [35] defined as below:

$$h_{i,\tau}^{(l)} = \mathrm{ReLU}(W^{(l)} \textstyle\sum_{(j,\tau') \in N_{i,\tau}} w_{i,\tau}(j,\tau')(h_{j,\tau'}^{(l-1)} \parallel e_{ij}(\tau') \parallel \phi(t-\tau'))) \tag{9}$$

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 [35] or learn the weights using attention modules [53].

3.2 Influence Analysis

Key Intuition. We observe that the weight $w_{i,\tau}(j,\tau')$ used in the generic aggregator (Eq.9) implicitly define the 1-step temporal random walk probability $P_{i,\tau}(j,\tau')$ from (i,τ) to (j,τ'). 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 theoretical analysis shows 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.

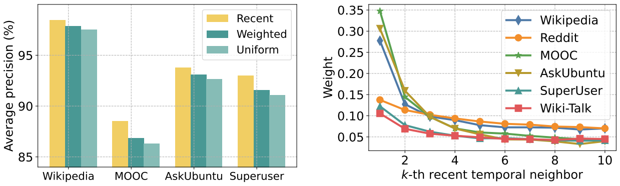
Theorem 2. *For an L-layer T-GNN with the generic aggregator (Eq.9), the normalized influence score $I_{i,t}(s_j)$ is proportional to the L-step temporal random walk probability from (i,t) to the node v_j.*

Proof. Denote by R a set of paths from (i,t) to node v_j in the computation graph of T-GNN. Each path $r \in R$ can be represented by a sequence of temporal nodes $(u_L,\tau_L),\ldots,(u_1,\tau_1),(u_0,\tau_0)$, where $(u_L,\tau_L) = (i,t)$ and $u_0 = j$. Let $[\partial h_{i,t}^{(L)}/\partial s_j]_r$ be the partial derivative computed along the path r, then $\partial h_{i,t}^{(L)}/\partial s_j = \sum_{r \in R}[\partial h_{i,t}^{(L)}/\partial s_j]_r$.

We decompose the weight matrix $W^{(l)}$ into $W^{(l)} = W_h^{(l)} \parallel W_e^{(l)} \parallel W_\phi^{(l)}$, where $W_h^{(l)}, W_e^{(l)}, W_\phi^{(l)}$ are the weights applied to intermediate embeddings, edge features, and time encodings, respectively. Let $f_{i,\tau}^{(l)}$ be the pre-activated feature, i.e., $h_{i,\tau}^{(l)}l = \mathrm{ReLU}(f_{i,\tau}^{(l)})$. By the chain rule, $[\partial h_{i,t}^{(L)}/\partial s_j]_r = \prod_{l=L}^{1} w_{u_l,\tau_l}(u_{l-1},\tau_{l-1})W_h^{(l)}\partial h_{i,\tau}^{(l)}/\partial f_{i,\tau}^{(l)}$. Here, the weight $w_{u_l,\tau_l}(u_{l-1},\tau_{l-1})$ corresponds to the 1-step temporal random walk probability from (u_l,τ_l) to (u_{l-1},τ_{l-1}).

We next analyze the derivative of the ReLU function $\partial h_{i,\tau}^{(l)}/\partial f_{i,\tau}^{(l)}$. For each entry in $[\partial h_{i,t}^{(L)}/\partial s_j]_r$, there can be multiple paths of neurons contributing to its value. However, some paths may be deactivated by the ReLU function. Here, we follow the randomization assumption [10, 18, 54] on the ReLU activations and assume that the probability of each path of neurons being activated is ρ. Hence, $E[\partial h_{i,t}^{(L)}/\partial s_j]_r = \rho \prod_{l=L}^{1} w_{u_l,\tau_l}(u_{l-1},\tau_{l-1})W_h^{(l)}$. Combine the partial derivative computed along each path $r \in R$, we have $E[\partial h_{i,t}^{(L)}/\partial s_j] = \sum_{r \in R} E[\partial h_{i,t}^{(L)}/\partial s_j]_r = \rho(\prod_{l=L}^{1} W_h^{(l)})P_{i,t}(j;L)$, where $P_{i,t}(j;L)$ is the L-step temporal random walk probability from (i,t) to v_j. □

Theorem 3. *For an L-layer T-GNN with the generic aggregator (Eq.9), the normalized influence score $I_{i,t}(e_{jz}(\tau))$ is proportional to a weighted sum of l-step ($1 \le l \le L$) temporal random walk probability from (i,t) to (z,τ).*



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

Figure 3: (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.

Proof. We prove Theorem 3 following the notations and basic idea used in the proof of Theorem 2. As shown in Figure 2a, unlike state vectors, edge features can participate in T-GNN computation at each intermediate layer. By the chain rule of differentiation and the randomization assumption on ReLU activations, $E[\partial h_{i,t}^{(L)}/\partial e_{jz}(\tau)] = \sum_{k=1}^{L} \rho(\prod_{l=L}^{L-k+2} W_h^{(l)})W_e^{(L-k+1)}P_{i,t}(z,\tau;k)$, where $P_{i,t}(z,\tau;k)$ is the k-step 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 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.

3.3 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 [41]. Figure 3a 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 [35]. Given a target node, the weighted sampling approach [49] assigns recently interacted neighbors higher probabilities following an exponential distribution; the recent sampling approach [35] simply selects the top-k recent temporal neighbors for feature aggregation. Figure 3a suggests 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 3b 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.

Based on the observations, we propose to utilize the T-PPR metric (as defined in Def. 2) to estimate the temporal random walk probabilities over dynamic graphs. Specifically, the T-PPR value $\pi_{i,t}(j,\tau)$ measures the probability that an α-temporal random walk with β-exponential decay starting from (i,t) terminates at (j,τ). Based on Theorem 2 and Theorem 3, $\pi_{i,t}(j,\tau)$ corresponds to the influence of 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 [9, 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.

4 THE ZEBRA FRAMEWORK

4.1 Framework Overview

In this section, we present Zebra, a framework that improves T-GNN via T-PPR. Figure 2b 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 5. Moreover, we empirically show that a small value of k can already lead to remarkable model performance in Section 6.
- **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 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 5. In contrast, as discussed in Section 2.4, 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.

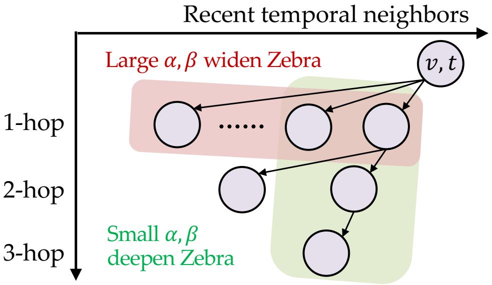


Figure 4: An illustration of the ensemble idea. The shades of red and green indicate the top-k influential temporal neighbors of (v,t) selected by two different T-PPR metrics. In theory, an ensemble of diversified T-PPR metrics can widen and deepen the receptive field of Zebra at the same time, thus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

4.2 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 [38, 60], 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:

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

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 6.

5 TOP-k T-PPR QUERY

Existing algorithms for PPR cannot be adapted to efficiently solve the top-k T-PPR query due to the *constantly changing graph structure* and the *highly imbalanced weight distribution.* Please refer to Section 7 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 5.1). We then propose a *single-scan* algorithm (Section 5.2) to answer the *up-to-date* top-k T-PPR query with rigorous approximation guarantees (Section 5.3).

5.1 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 7). 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 5.2.

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.

Proof. On the one hand, for any temporal neighbor $(z, \tau) \in N_{i,t^-}$, we can build the connection between $P_{i,t^+}(z, \tau)$ and $P_{i,t^-}(z, \tau)$:

$$P_{i,t^+}(z, \tau) = \frac{\beta^{|\{(z',\tau')|(z',\tau') \in N_{i,t^+}, \tau' \geq \tau\}|}}{\sum_{k=1}^{|N_{i,t^+}|} \beta^z} = \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta} P_{i,t^-}(z, \tau). \tag{14}$$

On the other hand, for the newly arrived temporal node (j, t), we have the following transition probability:

$$P_{i,t^+}(j, t) = \frac{\beta}{\sum_{z=1}^{|N_{i,t^+}|} \beta^z} = \frac{\beta}{m_{i,t^-}\beta+\beta}. \tag{15}$$

Based on Eq.14 and Eq.15, we know that

$$\begin{aligned} \pi_{i,t^+} &= \textstyle\sum_{(z,\tau) \in N_{i,t^+}} (1-\alpha) P_{i,t^+}(z, \tau)(\alpha I_{z,\tau} + \pi_{z,\tau}) \\ &= \textstyle\sum_{(z,\tau) \in N_{i,t^-}} (1-\alpha) P_{i,t^+}(z, \tau)(\alpha I_{z,\tau} + \pi_{z,\tau}) \\ &\quad + (1-\alpha) P_{i,t^+}(j, t)(\alpha I_{j,t} + \pi_{j,t}) \\ &= \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\pi_{i,t^-} + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}(\pi_{j,t^-} + \alpha I_{j,t}). \end{aligned} \tag{16}$$

Then, the proof of Theorem 4 is completed. □

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.

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
1 $\bar{\pi}_{i,t^+} \leftarrow \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\tilde{\pi}_{i,t^-} + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}(\tilde{\pi}_{j,t^-} + \alpha I_{j,t})$
2 $\tilde{\pi}_{i,t^+} \leftarrow$ temporal nodes in $\bar{\pi}_{i,t^+}$ with the top-$k$ largest
    estimated T-PPR values
3 if $v_i \neq v_j$ then
4     $\bar{\pi}_{j,t^+} \leftarrow \frac{m_{j,t^-}\beta}{m_{j,t^-}\beta+\beta}\tilde{\pi}_{j,t^-} + \frac{\beta(1-\alpha)}{m_{j,t^-}\beta+\beta}(\tilde{\pi}_{i,t^-} + \alpha I_{i,t})$
5     $\tilde{\pi}_{j,t^+} \leftarrow$ temporal nodes in $\bar{\pi}_{j,t^+}$ with the top-$k$ largest
        estimated T-PPR values
6     $m_{j,t^+} \leftarrow m_{j,t^-}\beta + \beta$
7 $m_{i,t^+} \leftarrow m_{i,t^-}\beta + \beta$
8 return $\tilde{\pi}_{i,t^+}, \tilde{\pi}_{j,t^+}$
```

5.2 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.2).
- **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.2 (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 [23, 35, 53], 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.

5.3 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.

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,*

$$\tilde{\theta}^{(k)}_{i,t^+} \geq \max(\frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\tilde{\theta}^{(k)}_{i,t^-}, \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}\tilde{\theta}^{(k)}_{j,t^-}), \tag{17}$$

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

Proof. Denote by $|\tilde{\pi}_{i,t}|$ the number of elements in $\tilde{\pi}_{i,t}$, and let $\tilde{\theta}^{(k)}_{i,t} = 0$ if $|\tilde{\pi}_{i,t}| < k$. We start by proving that $\tilde{\theta}^{(k)}_{i,t^+} \geq \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\tilde{\theta}^{(k)}_{i,t^-}$. If $|\tilde{\pi}_{i,t^-}| < k$, the inequality holds since $\tilde{\theta}^{(k)}_{i,t^-} = 0$. If $|\tilde{\pi}_{i,t^-}| = k$, SANTA only keeps the top-k temporal nodes after weighted sum (lines 1-2). Hence, the inequality still holds. Similarly, we can prove that $\tilde{\theta}^{(k)}_{i,t^+} \geq \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}\tilde{\theta}^{(k)}_{j,t^-}$. Then, the proof is completed. □

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,*

$$\tilde{\pi}_{i,t^+}(z, \tau) \geq \max_{r \in R} P(r), \tag{18}$$

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

Proof. Denote by $r_{max} = \mathrm{argmax}_{r \in R} P(r)$ the path with maximum probability, and let r_{max} be represented by a sequence of temporal nodes $(i, t^+), \cdots, (z, \tau)$. SANTA may omit some temporal random walk paths when updating T-PPR dictionaries upon new interactions. If r_{max} contributes to $\tilde{\pi}_{i,t^+}(z, \tau)$, then Eq.18 holds. Otherwise, there must exist a temporal node (v', t') $(t' > \tau)$ in the sequence such that $\tilde{\theta}^{(k)}_{v',t'} > P((v', t') \longrightarrow (z, \tau)|r_{max})$, where the right-hand side denotes the probability of the partial path from (v', t') to (z, τ) along r_{max}. According to Lemma 5, as new interactions arrive, the probability of a temporal random walk path decreases faster than the top-k T-PPR threshold. Hence, $\tilde{\theta}^{(k)}_{i,t^+} \geq P(r_{max})$. Since $\tilde{\pi}_{i,t^+}(z, \tau) \geq \tilde{\theta}^{(k)}_{i,t^+}$, then $\tilde{\pi}_{i,t^+}(z, \tau) \geq P(r_{max})$. □

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*

$$\tilde{\pi}_{i,t^+}(z, \tau) \leq \pi_{i,t^+}(z, \tau) \leq \tilde{\pi}_{i,t^+}(z, \tau) + \frac{\theta^{(k)}_{max}}{\alpha(1-\beta)}, \tag{19}$$

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

Proof. Assume that for any (z, τ), there is a constant additive error bound λ $(\lambda > 0)$ on the estimation error introduced by SANTA. Then, $\pi_{i,t^-}(z, \tau) \leq \tilde{\pi}_{i,t^-}(z, \tau) + \lambda$ and $\pi_{j,t^-}(z, \tau) \leq \tilde{\pi}_{j,t^-}(z, \tau) + \lambda$. We next proceed by weak induction and study the following cases.

- If $(z, \tau) = (i, t)$, then there is no estimation error, i.e., $\pi_{i,t^+}(z, \tau) - \tilde{\pi}_{i,t^+}(z, \tau) = 0 < \lambda$. The assumption holds.
- If $(z, \tau) \neq (i, t)$ and (z, τ) is a top-k item in $\tilde{\pi}_{i,t^+}$, then $\pi_{i,t^+}(z, \tau) - \tilde{\pi}_{i,t^+}(z, \tau) \leq \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\lambda + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}\lambda \leq \lambda$. The assumption holds.
- If (z, τ) is not a top-k item in $\tilde{\pi}_{i,t^+}, \tilde{\pi}_{i,t^-}, \tilde{\pi}_{j,t^-}$, then $\pi_{i,t^+}(z, \tau) - \tilde{\pi}_{i,t^+}(z, \tau) \leq \frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\lambda + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}\lambda \leq \lambda$. The assumption holds.
- If (z, τ) is neither a top-k item in $\tilde{\pi}_{i,t^+}$ nor in $\tilde{\pi}_{i,t^-}$, but a top-k item in $\tilde{\pi}_{j,t^-}$, then $\frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}\tilde{\pi}_{j,t^-}(z, \tau) \leq \tilde{\theta}^{(k)}_{i,t^+}$. Then,

  $$\begin{aligned} &\pi_{i,t^+}(z, \tau) - \tilde{\pi}_{i,t^+}(z, \tau) = \pi_{i,t^+}(z, \tau) \\ \leq &\frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\lambda + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}(\lambda + \tilde{\pi}_{j,t^-}(z, \tau)) \\ \leq &\frac{m_{i,t^-}\beta}{m_{i,t^-}\beta+\beta}\lambda + \frac{\beta(1-\alpha)}{m_{i,t^-}\beta+\beta}\lambda + \tilde{\theta}^{(k)}_{i,t^+}. \end{aligned} \tag{20}$$

  Let the right-hand side in Eq.20 be bounded by λ. Then, the additive error assumption holds for $\tilde{\pi}_{i,t^+}$ if $\lambda \geq \frac{m_{i,t^-}\beta+\beta}{\alpha\beta}\theta^{(k)}_{i,t^+}$. To make the assumption holds for all the possible situations, we study the worst-case guarantee as follows:

  $$\frac{m_{i,t^-}\beta+\beta}{\alpha\beta}\theta^{(k)}_{i,t^+} \leq \frac{\frac{\beta}{1-\beta}}{\alpha\beta}\theta^{(k)}_{i,t^+} \leq \frac{\theta^{(k)}_{max}}{\alpha(1-\beta)}, \tag{21}$$

  where $\theta^{(k)}_{max}$ is the maximum possible top-k threshold. Therefore, the assumption holds if $\lambda \geq \frac{\theta^{(k)}_{max}}{\alpha(1-\beta)}$.
- If (z, τ) is neither a top-k item in $\tilde{\pi}_{i,t^+}$ nor in $\tilde{\pi}_{j,t^-}$, but a top-k item in $\tilde{\pi}_{i,t^-}$, then the addition error assumption holds if $\lambda \geq \frac{\theta^{(k)}_{max}}{\alpha(1-\beta)}$. Here, we skip the detailed proof, which is similar to the fourth case described above.

Summarizing the five cases above, then the proof is completed. □

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*

$$\pi_{i,t^+}(z, \tau) \leq \max(\frac{\alpha(1-\alpha)(1+\beta-\alpha)}{(1+\beta)\tilde{\theta}^{(k)}_{i,t^+}}, 1)\tilde{\pi}_{i,t^+}(z, \tau) \tag{22}$$

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

Proof. Denote by R a set of temporal random walk paths that start from (i, t^+) and stop at (z, τ). Then, $\pi_{i,t^+}(z, \tau) = \sum_{r \in R} P(r)$, where $P(r)$ denotes the probability of taking the path r. Clearly, there is no estimation error, i.e., $\pi_{i,t^+}(z,\tau) = \tilde{\pi}_{i,t^+}(z,\tau)$ if $|R| = 1$. Moreover, we have the following upper bound on the relative error:

$$\frac{\pi_{i,t^+}(z,\tau)}{\tilde{\pi}_{i,t^+}(z,\tau)} = \frac{\sum_{r\in R} P(r)}{\tilde{\pi}_{i,t^+}(z,\tau)} \leq \frac{\sum_{r\in R} P(r)}{\tilde{\theta}^{(k)}_{i,t^+}}. \tag{23}$$

We next bound the value of $\sum_{r\in R} P(r)$. By the definition of T-PPR, $\pi_{i,t^+}(z,\tau)$ achieves the largest value when (z,τ) is the only temporal neighbor of (i,t^+). In this case, $\pi_{i,t^+}(z,\tau) = (1-\alpha)\alpha$. According to Theorem 4, the probabilities of existing temporal random walk paths are weighted by factor smaller than one whenever a new interaction arrives. Therefore, increasing the number of temporal random walk paths from (i,t^+) to (z,τ) could inversely reduce the value of $\pi_{i,t^+}(z,\tau)$. For the case $|R| \geq 2$, $\pi_{i,t^+}(z,\tau)$ achieves the largest value $\frac{\alpha(1-\alpha)(1+\beta-\alpha)}{(1+\beta)}$ when $z = i$, and node v_i only has two self-loops. Then, the proof is completed. □

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.

6 EXPERIMENTS

In this section, we examine the efficiency and effectiveness of Zebra on real-world dynamic graphs (Section 6.2) and answer the following questions: (1) How do T-PPR parameters affect the performance of Zebra (Section 6.3)? (2) How does the performance of Zebra change with the ensemble size (Section 6.4)? (3) How does SANTA perform compared to baselines (Section 6.5)? (4) How does Zebra perform in different downstream tasks (Section 6.6)? (5) Does the implication of each theorem derived in Section 5.3 hold?

6.1 Experimental Settings

Datasets. Table 3 summarizes the six real-world dynamic graphs used in our experiments. MOOC, Wikipedia, and Reddit are common benchmarks for evaluating T-GNNs [23, 35, 49, 53]. 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 in Table 3, we chronologically split it into training set (70%), validation set (15%), and test set (15%) as in [35, 53]. 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 (see Section 6.2 to Section 6.5).

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 [35].
- JODIE [23] uses RNNs to propagate the information of temporal interactions to update node representations.
- TGAT [53] mimics the message passing scheme of static GNNs and encodes time information through random Fourier features.
- TGN [35] dynamically maintains a state vector for each node and includes previous methods [23, 40, 53] as special cases.
- CAW [49] encodes temporal neighborhood information using anonymized random walk and attention modules.
- APAN [48] accelerates model inference by decoupling graph computation and message propagation.

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.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	d_v	d_e	Timespan
MOOC [23]	7,144	411,749	172	4	30 days
Wikipedia [23]	9,227	157,474	172	172	30 days
Reddit [23]	10,984	672,447	172	172	30 days
AskUbuntu [1]	159,316	964,437	172	0	2613 days
SuperUser [2]	194,085	1,443,339	172	0	2773 days
Wiki-Talk [3]	1,140,149	7,833,140	172	0	2320 days

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.3) 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 [35]. We have fixed the bugs in our empirical evaluation for a fair comparison.

Evaluation Metrics. Following prior works [35, 48, 49], we evaluate model performance in average precision (AP) and ROC AUC on the test set. 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:

$$\frac{\mathsf{sum}(\pi_{i,t}) - \mathsf{sum}(\tilde{\pi}_{i,t})}{\mathsf{sum}(\pi_{i,t})}, \tag{24}$$

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 [35, 49, 53]. 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 $\{1, 0.1, 0.01, 0.001\} \times$

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.

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.

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)**

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.

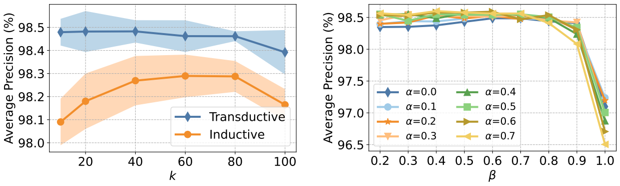
6.2 Key Results

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 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 [23]. TGAT and TGN follow the generic T-GNN architecture described in Section 2.4, thus suffering from high computational cost. Particularly, TGN is especially slow in updating the state vectors of nodes (Eq.6) 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.



(a) Impact of k **(b) Impact of α and β**

Figure 5: Performance of Zebra, equipped with a single T-PPR, on the Wikipedia dataset.

We omit the results due to the page limit. As have discussed in Section 3.3, Zebra is more effective for two reasons. First, our proposed T-PPR is capable of capturing the relative influence of multi-hop 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 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.

6.3 Analysis of T-PPR Parameters

Impact of k. Figure 5a demonstrates the average precision of Zebra achieved with a single T-PPR parameterized by $\alpha = 0.1$ and $\beta = 0.5$. There is a clear performance drop in both transductive and inductive settings when increasing k from 80 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. Moreover, Zebra can achieve superior performance in the transductive setting with a small k (e.g., 10 and 20), while it requires a larger k (e.g., 60) to achieve the best inductive learning ability. This is because inductive tasks 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. Therefore, a small neighborhood size k can work well in practice.

Impact of α, β. Figure 5b examines the effect of parameters α and β on Zebra's transductive performance. Clearly, Zebra is not sensitive to the choice of parameter α. In contrast, the configuration of β can have a major impact on model performance. Specifically, model performance remains relatively stable as we vary the value of β from 0.2 to 0.9. However, Zebra suffers from poor performance if we set $\beta = 1$. In this case, T-PPR degenerates into PPR, which assigns different temporal neighbors uniform weights. The above observation validates the critical intuition behind our proposed T-PPR, i.e., it would be beneficial to distinguish the influence of different temporal interactions via an exponential distribution.

Table 6: Performance of Zebra with $k = 20$ varying the ensemble size. 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	94.14	97.76	81.71

6.4 Analysis of Ensemble Size

Impact of Ensemble Size. 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 can achieve better performance on both Wikipedia and AskUbuntu datasets. 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 model performance [60] due to two reasons. The noises lead to the over-smoothing issue [9] and prevent the model from focusing on the most informative temporal information. Second, we train an MLP [34] 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 building an ensemble 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.

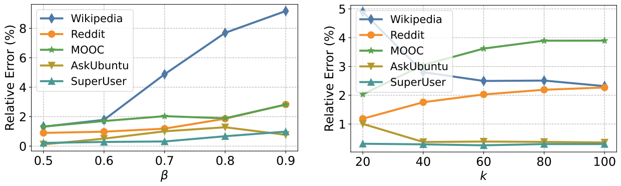
Robustness of Zebra. Another major advantage of the ensemble scheme is that it enhances the robustness of Zebra and frees practitioners from tedious parameter tuning. As shown in Figure 3b, the weights of the top-10 recent temporal neighbors learned by vanilla T-GNN approximately follow various exponential 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, as shown in Table 4, Table 5, and Table 6, 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 shown 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.

6.5 Analysis of SANTA

Superiority of SANTA. Table 7 shows the advantage of SANTA over the baseline PPI in efficiency and effectiveness. PPI is short for pruned power iteration, which mimics the recursive neighbor sampling strategy (Eq.3) adopted by vanilla T-GNNs. Specifically,

Table 7: Training Zebra with various top-20 T-PPR query algorithms on Reddit. The used T-PPR is configured by $\alpha = 0.1$ and $\beta = 0.5$. We report the query overhead (s), relative error (%) of query results, and transductive AP (%) and inductive AP (%) of trained models.

Algorithm	Overhead	Error	Trans AP	Induct AP
PPI(10, 2)	32.70	16.96	98.61	97.24
PPI(20, 2)	252.87	16.95	98.60	97.24
PPI(20, 3)	1134.43	4.16	98.62	97.29
SANTA	9.75	0.80	98.62	97.64



(a) Impact of β **(b) Impact of k**

Figure 6: (1) The relative error of SANTA in estimating top-20 T-PPR values with $\alpha = 0.1$. (2) The relative error of SANTA in estimating top-k T-PPR values with $\alpha = 0.1$ and $\beta = 0.7$.

PPI(w, d) recursively searches d-hop temporal information of a target temporal node and selects the top-w recent temporal neighbors at each step. Notably, SANTA is 116 times faster than PPI(20, 3) while attaining significantly smaller relative error (i.e., 0.8%). Moreover, SANTA leads to higher inductive AP. As discussed in Section 6.2, this is because T-PPR enlarges the receptive field of Zebra regardless of the number of hops, while classical solutions like PPI suffers from limited receptive field.

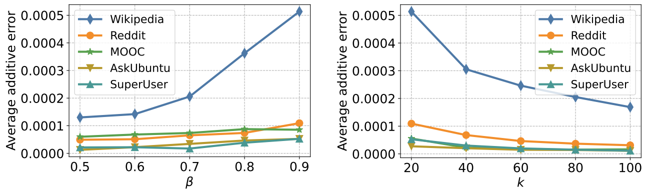
Impact of Graph Distribution. Figure 6 examines the relative error (Eq.24) introduced by SANTA in estimating top-k T-PPR values. Notably, the relative error is marginal (i.e., within 9%) on various graph distributions. The results suggest that SANTA can effectively capture most of the essential temporal information over dynamic graphs. Besides, the relative error generally decreases as the temporal random walk probabilities become more skewed (i.e., the value of β adopted by T-PPR becomes small). This observation is consistent with our theoretical analysis in Theorem 7. Moreover, Figure 6b shows that the error introduced by SANTA remains small as we vary the value of k.

Table 8: ROC AUC (%) for dynamic node classification.

Dataset	JODIE	TGAT	TGN	APAN	Zebra
Wikipedia	84.84	83.69	87.81	88.72	**89.12**
Reddit	61.83	65.56	**67.06**	65.24	66.95

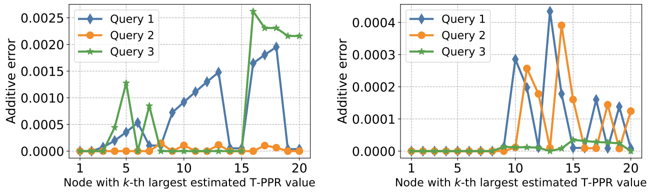
6.6 Evaluation on Dynamic Node Classification

Existing works [35, 48, 49, 53] mainly consider link prediction tasks. We additionally evaluate the performance of Zebra on two node classification tasks, i.e., Wikipedia and Reddit. These two datasets contain node labels indicating whether a user is banned. As other datasets in Table 3 do not contain labels, they cannot be used as node classification tasks. Table 8 shows that compared to the baselines, Zebra can also achieve remarkable performance in dynamic node classification. We omit the comparison of training and inference time since Zebra can achieve a similar speedup as shown in Section 6.2. Overall, the results suggest that Zebra can achieve robust performance in various downstream tasks.



(a) Impact of β **(b) Impact of k**

Figure 7: Justification for the implication of Theorem 7. We show the average additive errors introduced by SANTA in answering top-k queries on five datasets.



(a) Wikipedia **(b) AskUbuntu**

Figure 8: Justification for the implication of Theorem 8. We show the additive errors introduced by SANTA in answering three randomly picked top-20 T-PPR queries on the Wikipedia and AskUbuntu datasets, respectively.

6.7 Validation of Theorems in Section 5.3

Validation of Theorem 7. Figure 7 shows the average additive error in answering top-k T-PPR queries on five datasets. By default, we parameterize the T-PPR metric with $\alpha = 0.1$, $\beta = 0.9$, and $k = 20$. As the value of β decreases, the temporal random walk probabilities become more skewed. Meanwhile, as shown in Figure 7a, the average additive error for each node generally increases as the value of β decreases. Besides, Figure 7b demonstrates that the average additive error introduced by SANTA clearly decreases as the value of k increases. The above two observations validate the implication of Theorem 7 that the additive errors are negligible when temporal random walk probabilities are skewed or a large k is adopted.

Validation of Theorem 8. Figure 8 shows the additive errors in answering three randomly picked top-20 T-PPR queries on the Wikipedia and AskUbuntu datasets. The additive error is marginal or even zero for a few most influential temporal nodes. This phenomenon is particularly evident on the AskUbuntu dataset. On the other hand, the additive error tends to be relatively larger for the less influential temporal nodes. Overall, the above observations validate the implication of Theorem 8 that the estimation errors can be marginal for the most influential temporal nodes.

6.8 Guidance on the choice of hyperparameters

In this section, we provide guidance on the choice of hyperparameters. Please see our technical report[1] for the 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.

[1] https://github.com/LuckyLYM/Zebra/technical_report

Parameters α and β. Zebra is not sensitive to the choice of parameter α. We empirically observe that the best setting for parameter α generally lies in the range of $[0.3, 0.5]$. Besides, 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.

Neighborhood size k. The choice of neighborhood size k depends on the connectivity of a given node. In general, Zebra, equipped with a single T-PPR, 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. To achieve robust performance, we recommend implementing Zebra with $k = 20$ over an ensemble of two diversified T-PPR metrics.

Ensemble size n. We recommend setting the ensemble size $n = 2$ in practice. On the one hand, a single T-PPR metric, if not well parameterized, may fail to sufficiently capture the diffusion process on evolving graphs, thus degrading model accuracy. Increasing the ensemble size may not lead to better performance. This is because a large ensemble size could introduce noises, insignificant neighbors, extra parameters, and tuning efforts.

6.9 Summary

We summarize the key experimental findings as follows.

- Zebra is one or two orders of magnitude faster than the state-of-the-art T-GNNs while attaining better model performance (Table 4 and Table 5). Moreover, Zebra is particularly effective in the inductive setting as our T-PPR enlarges the receptive field of T-GNN irrespective of the number of hops.
- Although SANTA requires extra space for dynamically maintaining approximated top-k T-PPR dictionaries, Zebra is still memory efficient (Table 5). Zebra prunes the computational graph of vanilla T-GNNs, thus saving the GPU memory required to finish the corresponding computations.
- In practice, Zebra implemented as an ensemble of two distinct T-PPR metrics demonstrates robust and superior performance on various dynamic graphs (Table 4, Table 5, and Table 6).
- SANTA is significantly more efficient than classical recursive neighbor sampling strategy adopted by vanilla T-GNNs while incurring less approximation errors (Table 7).

7 RELATED WORK

Temporal Graph Neural Networks. T-GNNs are powerful models for learning representations on continuous-time dynamic graphs. The pioneering work [23] uses a time projection module to predict future embeddings and adopts RNNs to update state vectors of nodes. DyRep [40] further incorporates 2-hop neighborhood when updating node representations. TGAT [53] mimics the message passing scheme of static GNNs and encodes time information through random Fourier features. TGN [35] unifies previous methods [23, 40, 53] and have achieved remarkable accuracy improvement. CAW [49] encodes sampled random walks using an anonymization strategy to further improve the predictive performance on unseen nodes. Moreover, APAN [48] 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 [20], which derives a new propagation scheme based on personalized PageRank (PPR). The follow-up work [7] further accelerates approximated PPR computation by parallelizing the forward push algorithm [5]. Other graph diffusion methods like heat kernel PageRank [21] and ordinary differential equation [8] are also explored in GNN computation. Moreover, the randomized algorithm AGP [42] provides a unified solution for approximately computing various proximity queries. To build more robust and power GNNs, SIGN [36] and GMLP [57] combines various diffusion methods for information propagation. In contrast to prior works, we are the first to study the diffusion process for T-GNNs. We have proposed a novel metric named T-PPR for T-GNNs and devised an efficient top-k T-PPR algorithm with approximation guarantees.

Personalized PageRank. PPR [32] is a critical proximity measure for static graphs. The advanced algorithms [26, 28, 45–47, 51] for fast PPR computation are the combination of basic techniques including forward push [5], reverse push [4], power iteration [32], and Monte-Carlo sampling. For instance, a line of research [26, 46, 47] 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 [6, 29, 55, 56] incrementally update indexes (e.g., random walk segments) when the graph structure changes. As for top-k PPR query, early works [11, 12, 52] 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 [39, 50] 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 [4, 5, 46] and iteration-based [26, 51] methods would spend significant time on calculating a tiny probability mass, resulting in severe overhead. While the EdgePush algorithm [44] can address this issue, it is not applicable to dynamic graphs. Unlike SANTA, EdgePush needs to precompute and maintain a threshold for each edge in the graph.

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.

REFERENCES

[1] [n.d.]. AskUbuntu. http://snap.stanford.edu/data/sx-askubuntu.html.

[2] [n.d.]. SuperUser. http://snap.stanford.edu/data/sx-superuser.html.

[3] [n.d.]. wiki-talk. http://snap.stanford.edu/data/wiki-talk-temporal.html.

[4] 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.

[5] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *FOCS*. IEEE Computer Society, 475–486.

[6] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *PVLDB* 4, 3 (2010), 173–184.

[7] 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.

[8] 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.

[9] 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.

[10] Anna Choromanska, Yann LeCun, and Gérard Ben Arous. 2015. Open Problem: The landscape of the loss surfaces of multilayer networks. In *COLT (JMLR Workshop and Conference Proceedings)*, Vol. 40. JMLR.org, 1756–1760.

[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] Kenji Kawaguchi. 2016. Deep Learning without Poor Local Minima. In *NIPS*. 586–594.

[19] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *CoRR* abs/1611.07308 (2016).

[20] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*. OpenReview.net.

[21] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *NeurIPS*. 13333–13345.

[22] 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.

[23] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *SIGKDD*. ACM, 1269–1278.

[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] Allan Pinkus. 1999. Approximation theory of the MLP model in neural networks. *Acta numerica* 8 (1999), 143–195.

[35] 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).

[36] 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

[37] Polina Rozenshtein and Aristides Gionis. 2016. Temporal PageRank. In *ECML (Lecture Notes in Computer Science)*, Vol. 9852. Springer, 674–689.

[38] Omer Sagi and Lior Rokach. 2018. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 4 (2018).

[39] 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.

[40] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.

[41] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[42] 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.

[43] Hongwei Wang and Jure Leskovec. 2020. Unifying Graph Convolutional Neural Networks and Label Propagation. *CoRR* abs/2002.06755 (2020).

[44] 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).

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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*.

[50] 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.

[51] 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.

[52] 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.

[53] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. In *ICLR*.

[54] 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.

[55] 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.

[56] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate Personalized PageRank on Dynamic Graphs. In *KDD*. ACM, 1315–1324.

[57] 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).

[58] 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.

Table 9: Summary of the data statistics. $|E|$, $|U|$, and $|I|$ indicate the number of temporal interactions, users, and items, 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 $d = \sum_{u \in U} d_{u,\infty}/|U|$.

Dataset	$\|E\|$	$\|U\|$	$\|I\|$	d (diversity)
Wikipedia	157,474	8,227	1,000	1.66
Reddit	672,447	1,000	984	6.49
MOOC	411,749	7,047	97	1.86
AskUbuntu	964,437	120,818	127,047	1.46

[59] 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.

[60] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms*. Vol. 14.

A APPENDIX

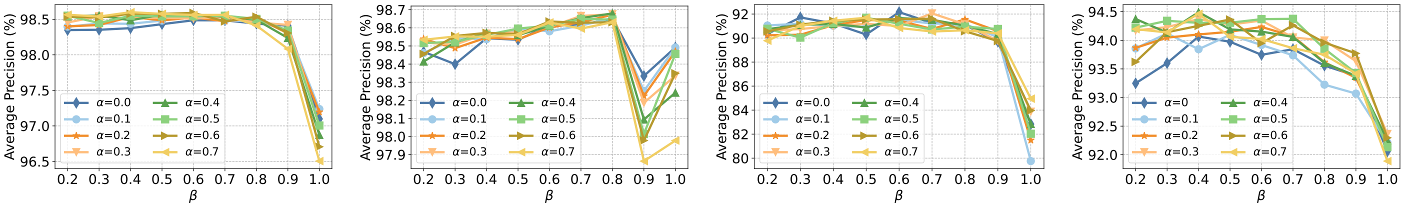
A.1 Guidance on the choice of hyperparameters

In this section, we conduct extensive experiments to provide guidance on the choice of hyperparameters. Table 9 summarizes the statistics of dynamic graphs used in our experiments. "diversity" [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.

Based on the above preliminaries, we summarize the key points as follows. 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 dynamic graph.

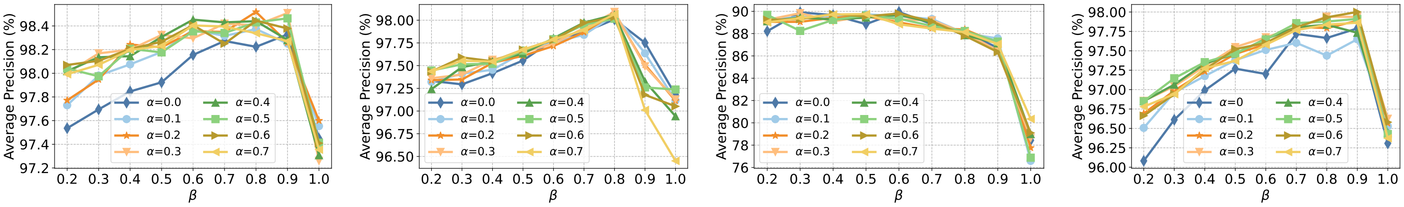
- **Parameter β.** As shown in Figure 4, a smaller β value makes T-PPR perceive a smaller neighborhood, while a larger β value makes T-PPR perceive a larger neighborhood. In general, we suggest parameterizing a single T-PPR with a small β value (e.g., 0.5) to learn embeddings for transductive nodes with low-diversity interaction patterns. Otherwise, a large β value (e.g., 0.9) is preferred. The above claims are supported by our new experimental results shown in Figure 9 and Figure 10.
  - Figure 9 investigates the effect of parameter β on Zebra's transductive average precision. It requires a small β to achieve the best transductive performance on Wikipedia, MOOC, and AskUbuntu. As shown in Table 9, the three dynamic graphs have low-diversity and easy-to-predict interaction patterns. In other words, a target node in these graphs 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 lead to remarkable transductive performance. Increasing the β value may incorporate insignificant and noisy neighbors, thus deteriorating transductive performance for the three dynamic graphs. 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 10 further investigates the effect of parameter β on Zebra's inductive average precision. Clearly, Wikipedia, 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. Moreover, notice that the best setting for parameter β is exceptional on MOOC, where a small β value (e.g., 0.5) leads to better inductive performance. MOOC is a small and densely-connected dynamic graph with 411,749 temporal interactions and only 97 items. A small β value is enough to capture a given user node's neighborhood information for such small-scale dynamic graphs.

  Figure 11 further shows an ensemble of two diversified T-PPR metrics (e.g., parameterized by $\beta = 0.5$ and $\beta = 0.95$, respectively) demonstrates robust and remarkable performance in both transductive and inductive settings. This is because two diversified T-PPR metrics prioritize distinct temporal neighborhoods and generate complementary embeddings. More importantly, constructing an ensemble of such two T-PPR metrics frees practitioners from tedious parameter tuning.
- **Parameter α.** As shown in Figure 4, 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 not sensitive to the choice of parameter α. Figure 9 and Figure 10 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 the Wikipedia and AskUbuntu datasets. As suggested by Table 9, these two graphs have 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 may 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.
- **Neighborhood size k.** As shown in Figure 12 and Figure 13, the choice of neighborhood size k depends on the connectivity of a given node. In general, Zebra, equipped with a single T-PPR, 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 test model performance on nodes that are not observed during model training. Given a newly appeared node, it generally needs to take a sufficiently large neighborhood to learn its temporal representation. In contrast, the state vector of a transductive node may have already captured enough information about past temporal interactions. Therefore, a small neighborhood size k can work well in practice. To achieve robust performance, we recommend implementing Zebra with $k = 20$



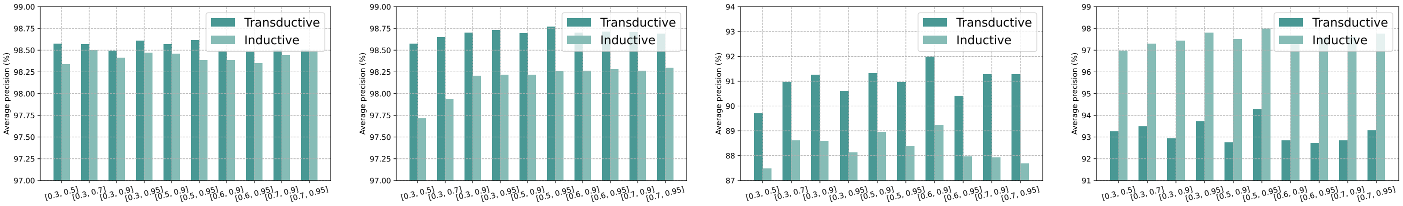
(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 9: Transductive performance of a single top-20 T-PPR varying parameters α and β.



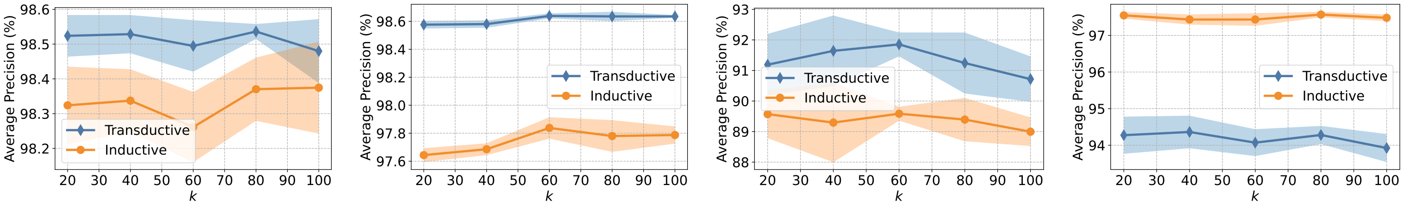
(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 10: Inductive performance of a single top-20 T-PPR varying parameters α and β.



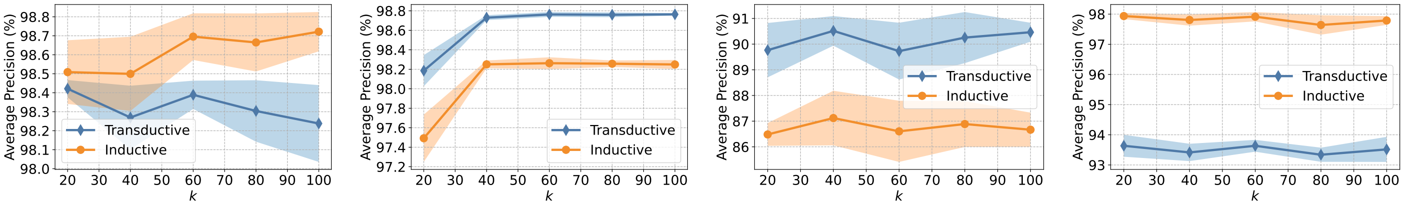
(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 11: Performance of an ensemble of two top-20 T-PPR metrics, parameterized by $\alpha = 0.3$, varying parameters β. The abscissa shows the β values used in two base T-PPR metrics.



(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 12: Performance of a single T-PPR, parameterized by $\alpha = 0.3$ and $\beta = 0.5$, varying neighborhood size k.

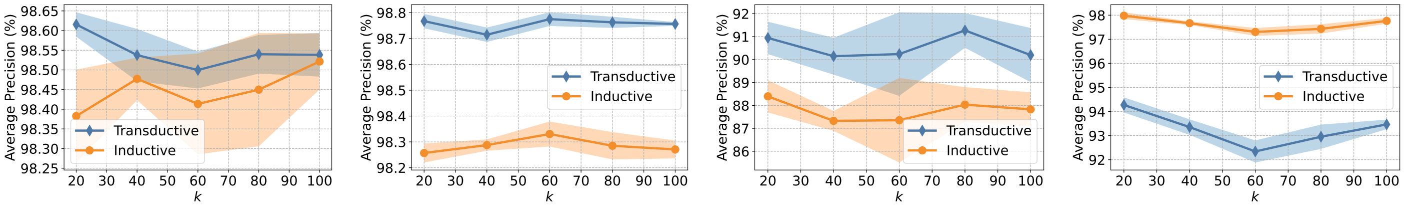


(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 13: Performance of a single T-PPR, parameterized by $\alpha = 0.3$ and $\beta = 0.9$, varying neighborhood size k.

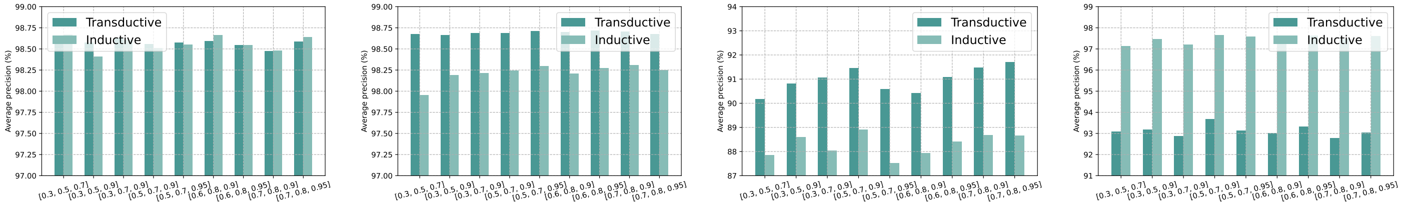
over an ensemble of two diversified T-PPR metrics. Figure 14 show that this parameter setting $k = 20$ is effective on a wide range of datasets. On the one hand, a small value of $k = 20$ would rule out noises introduced by insignificant nodes so that Zebra can perform well in the transductive setting. On the other hand, an ensemble of two diversified top-20 T-PPR metrics can widen and deepen the receptive field of Zebra at the same time so that Zebra can perform well in the inductive setting. In addition, setting a small neighborhood size k also reduces the time complexity of Zebra and thus brings more efficiency gains.

- **Ensemble size n.** We recommend setting the ensemble size $n = 2$ in practice. Figure 9 and Figure 10 demonstrate Zebra,



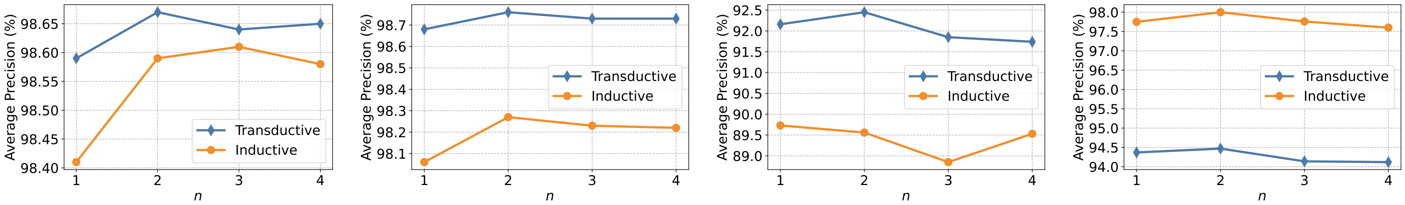
(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 14: Performance of an ensemble of two top-20 T-PPR metrics, parameterized by ($\alpha = 0.3, \beta = 0.5$) and ($\alpha = 0.3, \beta = 0.95$), varying neighborhood size k.



(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 15: Performance of an ensemble of three top-20 T-PPR metrics, parameterized by $\alpha = 0.3$, varying parameters β. The abscissa shows the β values used in two base T-PPR metrics.



(a) Wikipedia **(b) Reddit** **(c) MOOC** **(d) AskUbuntu**

Figure 16: Performance of Zebra varying ensemble size n. We report the best tuned results for each configuration of ensemble size n.

equipped with a single T-PPR, can already achieve remarkable performance in some cases. However, a single T-PPR metric, if not well parameterized, may fail to sufficiently capture the diffusion process on evolving graphs, thus degrading model accuracy. Fortunately, we can build an ensemble of multiple T-PPR metrics to address the above limitation. In principle, ensemble learning is more pronounced if base models are diverse [60]. As shown in Figure 4, two diversified T-PPR metrics prioritize distinct temporal neighborhoods and generate complementary temporal embeddings. Hence, an ensemble of such two diversified T-PPR metrics (e.g., parameterized by $\beta = 0.5$ and $\beta = 0.95$, respectively) can enhance model accuracy and robustness. On the other hand, comparing Figure 11 and Figure 15, it is clear that increasing the ensemble size from $n = 2$ to $n = 3$ may not lead to better performance. Besides, Figure 4 demonstrates that a small ensemble size $n = 2$ generally works well on all four compared dynamic graphs. This is because a large ensemble size could introduce noises, insignificant neighbors, extra parameters, and tuning efforts. Furthermore, increasing the ensemble size n introduces extra training time. Based on the above discussions, we suggest setting the ensemble size $n = 2$ in practice.