Scalable and Efficient Temporal Graph Representation Learning via Forward Recent Sampling

Yuhong Luo
Rutgers University
y.luo@rutgers.edu

Pan Li
Georgia Institute of Technology
panli@gatech.edu

Abstract

Temporal graph representation learning (TGRL) is essential for modeling dynamic systems in real-world networks. However, traditional TGRL methods, despite their effectiveness, often face significant computational challenges and inference delays due to the inefficient sampling of temporal neighbors. Conventional sampling methods typically involve backtracking through the interaction history of each node. In this paper, we propose a novel TGRL framework, No-Looking-Back (NLB), which overcomes these challenges by introducing a forward recent sampling strategy. This strategy eliminates the need to backtrack through historical interactions by utilizing a GPU-executable, size-constrained hash table for each node. The hash table records a down-sampled set of recent interactions, enabling rapid query responses with minimal inference latency. The maintenance of this hash table is highly efficient, operating with $O(1)$ complexity. Fully compatible with GPU processing, NLB maximizes programmability, parallelism, and power efficiency. Empirical evaluations demonstrate that NLB not only matches or surpasses state-of-the-art methods in accuracy for tasks like link prediction and node classification across six real-world datasets but also achieves 1.32-4.40$\times$ faster training, 1.2-7.94$\times$ greater energy efficiency, and 1.63-12.95$\times$ lower inference latency compared to competitive baselines.

The link to the code: https://github.com/Graph-COM/NLB.

1 Introduction

Temporal networks are extensively employed to model real-world complex systems. Such networks dynamically evolve as interactions occur or states change within their nodes. Learning node representations in these networks is pivotal, as they can be used in many downstream tasks such as monitoring or predicting changes in the temporal networks. Representative applications include forecasting interactions between node pairs for the tasks such as anomalous transaction detection in financial networks [1–3], recommendations in social networks [4] and in e-commerce systems [5]. Node representations can also be used to predict individual nodes' properties such as community detection [6–8] and fraudsters detection [9–12].

The interaction patterns surrounding a node often provide key indicators of its present state. Inspired by Graph Neural Networks (GNNs) [13, 14] that may effectively encode the neighborhood of a node in static graphs, researchers have developed models for temporal networks to incorporate both the structural and temporal aspects of a node's historical neighborhood. These methods can be collectively named as temporal graph representation learning (TGRL) [15–21].

Despite their effectiveness, previous TGRL methods generally require much more computational resources compared to their static graph counterparts. This is particularly challenging for large-scale applications that demand real-time inference. In such scenarios, reducing *inference latency* — the time from query arrival to prediction response — becomes critical to avoid performance bottlenecks.

We identify that for most state-of-the-art (SOTA) TGRL methods, the online collection of the information around a node's neighborhood is a major computational bottleneck. As a query comes

Y. Luo and P. Li, Scalable and Efficient Temporal Graph Representation Learning via Forward Recent Sampling. *Proceedings of the Third Learning on Graphs Conference (LoG 2024)*, PMLR 269, Virtual Event, November 26–29, 2024.

in, current methods have to backtrack in time, sample a subset of historical interactions around the node of interest, and aggregate the information from these down-sampled neighboring interactions for prediction. Sampling neighboring nodes is a common strategy in representation learning for large static graphs [14, 22–24]. However, this strategy faces challenges when applied to temporal networks, as they do not incorporate temporal information and may encounter high latency due to the enlarged sample space created by neighbors across various timestamps.

	forward sampling	recent sampling	GPU sampling	node-level tasks
TGAT [17]	✗	✓	✗	✓
TGN [15]	✗	✗	✗	✓
CAWN [16]	✗	✓	✗	✗
APAN [21]	✓	✗	✗	✓
NAT [18]	✓	✗	✓	✗
NLB (this work)	✓	✓	✓	✓

Table 1: Comparisons between several TGRL methods.

To address this, a strategy termed *recent sampling* has been proposed [16, 17], where more recent interactions are given higher weights in sampling, such as through a probability proportional to $\exp(-c\triangle t)$, with $c \geq 0$ and $\triangle t$ reflecting the time difference, indicating the recency of an interaction. While recent sampling can improve prediction performance, the process of calculating sampling weights and executing non-uniform sampling has high computational complexity $O(|\mathcal{N}_u^t|)$ where $|\mathcal{N}_u^t|$ denotes the size of the historical neighbors of node u till time t. This significantly slows down the system. Consequently, most TGRL methods [15, 21, 25] opt for either simple *uniform sampling* or a strategy named *truncation*, where only a fixed number of most recent neighbors are considered, to reduce the complexity. Even with such simplification, the inference latency may still be high because backtracking and sampling of these historical interactions are performed within CPUs instead of GPUs due to the inherent irregularity. A recent framework called TGL [25] reduced the issue via leveraging powerful multi-core CPUs complemented by meticulously crafted C++ parallelizable programming for fine-grained memory and thread management. However, multi-threading in CPUs is less power-efficient and may still suffer from longer latency than parallelism in GPUs.

In this paper, we propose ***No-Looking-Back*** (NLB), an efficient and scalable framework for TGRL that aims to improve training and inference latency, and power efficiency without compromising on prediction accuracy. It adopts a novel sampling strategy called *forward recent sampling* that allows getting around backtracking historical interactions while achieving the benefits of recent sampling. We compare NLB with existing methods in Table 1. Our key idea is to maintain a GPU-executable size-constrained hash table for each node that records a set of down-sampled recent interactions. These down-sampled interactions can be directly used to track neighbors and to generate response for upcoming queries with extremely low inference latency. Moreover, the maintenance of this table only requires $O(1)$ complexity to deal with the new link. Specifically, we leverage hash collision to insert the new link into this table randomly to simulate the strategy of recent sampling. Importantly, all of these operations are implementable with PyTorch and are fully compatible with GPU processing. This ensures not only maximized programmability and parallelism but also improved power efficiency.

NLB supports two variants NLB-edge and NLB-node, which differ in the used hash keys being either link ID or node ID. NLB-edge and NLB-node provably achieve recent sampling of links and recent sampling of nodes respectively while both support node-level and link-level applications. Empirically, NLB is comparable or improves SOTA link prediction and node classification accuracy in 6 real-world datasets while being 1.32-4.40× faster in training and 1.2-7.94× more energy efficient than the three most competitive baselines, and 1.63-12.95× more efficient in inference latency than the fastest baseline that leverages powerful multi-thread CPUs to perform backward sampling.

2 More Related Works

There are two streams of work on representation learning for temporal networks. Earlier work proposed aggregating the sequence of links into network snapshots and processing a sequence of static network snapshots [26–31], but they may suffer from low prediction accuracy as they cannot capture time-sensitive information within static snapshots. More recent temporal graph learning methods process link streams directly [19, 20, 32–35]. Most of these methods learn representation for a node using the historical information from the neighboring nodes it previously interacted with [15–18, 25, 36–38].

There are many works for graph learning that consider improvement on scalability and inference efficiency. For GNNs on static graphs, a stream of work focuses on efficient neighbor sampling [14, 22–24] which speeds up both training and inference, while some other works studies the acceleration for inference via knowledge distillation [39–41] or channel pruning [42]. However, these techniques cannot be easily generalized to improve the scalability for temporal networks. First, as temporal links come in as a sequence, it is non-scalable to construct and maintain the whole graph for every new

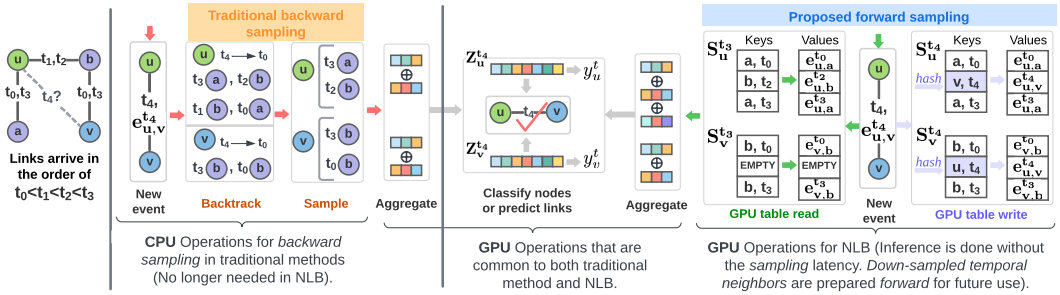

Figure 1: A toy example comparing the sampling done in current TGRL methods v.s. in NLB. *Left*: Given the toy historical temporal network, the task is either (1) to classify nodes u and v at time t_4, or (2) to predict whether u will connect with v at t_4. *Middle*: As a new link (u, v, t_4) arrives, traditional methods require looking backward and sampling a subset of historical temporal neighbors (via CPU). The generated embeddings of the temporal neighbors are aggregated to generate the final representations $Z_u^{t_4}$ and $Z_v^{t_4}$ for making predictions. *Right*: NLB abandons the backward sampling in traditional methods and adopts forward sampling. For inference, it directly looks up the down-sampled temporal neighbors from GPU hash tables for aggregation and prediction without being slowed down by the sampling operations. The new interaction can replace older temporal neighbors in the hash tables leveraging hash collision with complexity $O(1)$. The updated hash tables capture the new down-sampled temporal neighbors for use later on.

link. Furthermore, these methods do not resolve the bottleneck of historical neighbor sampling on temporal graphs, but neighborhood information is crucial for graph representation learning.

For temporal graphs, due to the more significant computation overhead of TGRLs as reviewed in Sec. 1, several recent works focusing on acceleration and efficiency inference are proposed [18, 21, 25, 43]. APAN [21] proposed asynchronous graph propagation which passes information to the most recent temporal neighbors and works with a sampling property similar to truncation without sampling. It improves inference latency but requires significant memory for buffering messages and the recent truncation strategy may hurt prediction accuracy. A recent work NAT [18] aims to improve link prediction performance and scalability by learning neighborhood-aware representations, which elegantly avoids backward sampling. However, its sampling property is not mathematically explained, and it relies on constructing joint neighborhood structural features of node pairs, which cannot be generalized to node-level tasks. In experiment (Sec. 5), we find that by removing the joint neighborhood features from NAT, it can be applied to node-level tasks. But its node-level prediction performance is worse than NLB and it is still slow compared to NLB.

Finally, temporal graph benchmark (TGB) [44] is proposed to standardize the evaluation pipelines for TGRL. We build a separate evaluation pipeline outside of TGB because our focus involves computational evaluations on billion-scale datasets which has not been supported by TGB. We provide supplementary evaluation of NLB on TGB in Appendix F.

3 Notations and Problem Formulation

Next, we introduce some notations and the problem formulation. We consider temporal network as a sequence of timestamped interactions between pairs of nodes.

Definition 3.1 (Temporal network)**.** A temporal network $\mathcal{E}$ can be represented as $\mathcal{E} = \{(u_1, v_1, t_1), (u_2, v_2, t_2), \cdots\}$, $t_1 \leq t_2 \leq \cdots$ where u_i and v_i denote interacting node IDs of the ith link, t_i denotes the timestamp. Each temporal link (u, v, t) may have link features $e_{u,v}^t$. We also denote the entire node set as $\mathcal{V}$. Without loss of generality, we use integers as node IDs, i.e., $\mathcal{V} = \{1, 2, ...\}$. For simplicity of the notations, we assume the links are undirected.

The dynamics of a node's neighborhood provide crucial information about a node current state. We define the temporal neighbors of a node as follows.

Definition 3.2 (Temporal neighbors of a node)**.** We denote the temporal neighbors of node u at time t as all neighbors interacted in the past. They are represented by tuples of opposing node IDs, link features and timestamps:

$$\mathcal{N}_u^t := \{(v, e_{u,v}^{t'}, t') \mid (u, v, t') \in \mathcal{E}, t' < t\}. \tag{1}$$

Finally, we formulate our problem as follows.

Definition 3.3 (Problem formulation)**.** Our problem is to learn a model that uses the historical information including the temporal neighbors about a node (e.g., u) before a timestamp (e.g., t) to efficiently generate a node representation that is predictive for downstream tasks including link prediction and node classification. We denote the representation at timestamp t for each node $u \in \mathcal{V}$ as Z_u^t. We define the link prediction and node classification problems as follows.

Link prediction: given any pair of representations, (Z_u^t, Z_v^t), a link prediction task predicts whether there will be a temporal link between the two nodes at time t, i.e., $(u, v, t) \in \mathcal{E}$. *Node classification:* a node u can be dynamically labeled, Let y_u^t denote the label of u at timestamp t. A node classification task predicts y_u^t given the representation Z_u^t.

4 Methodology

In this section, we first review the sampling strategies of previous approaches. We then introduce forward recent sampling, the sampling strategy of NLB and show that it provably achieves recent sampling. Lastly, we present the efficient aggregation of temporal neighbors and the generation of node presentations for inference.

4.1 The sampling strategies of existing works

While different approaches have different implementations, a common computational bottleneck for generating node representations during both training and inference is the online sampling of temporal neighbors. Existing sampling strategies consist of two steps: (1) Backtrack and collect the historical interactions of a node from the current timestamp; (2) Sample a subset of temporal neighbors from those interactions. Traditional methods [15–17, 25] perform these operations within CPUs, causing inefficiency in both time and power consumption for both training and inference.

To give an example, to learn the representations for node u and v at t_4 for downstream predictions (Fig. 1 Left), traditional methods perform backtracking and sampling within the CPUs (Fig. 1 Middle). These blocking operations not only slow down the training and inference processes, but also introduce costly energy consumption from both CPU computation and CPU-GPU communication. Previous methods commonly adopted the following sampling strategies.

Definition 4.1 (Truncation)**.** Truncation of a fixed number of most recent neighbors, or simply named *truncation*, is a common strategy used by TGN [15], APAN [21] and TGL [25]. It does not sample with randomness but truncates the most recent s temporal neighbors that appear in the history instead. Truncation is effective when the most recent neighbors provide sufficient information about the current state of a node's neighborhood. However, as it does not allow randomness, it may fail to capture information that appears earlier in the history. In an extreme, if all most recent interactions are from the same neighbor, truncation neglects information from all other early neighbors.

Definition 4.2 (Uniform sampling)**.** *Uniform sampling* is also commonly supported by TGN, TGL, TGAT [17], etc. It samples among all temporal neighbors in the past with equal probabilities. It is effective in capturing early-day neighbors which may benefit prediction. However, for many real-world applications, more recent events typically play a more important role. Moreover, uniform sampling, albeit straightforward, still requires the access to all of the historical temporal interactions. Since different nodes have different numbers of historical interactions, uniform sampling can hardly be implemented in the GPUs for parallelism, and thus is slower than truncation.

4.2 Our approach: Forward recent sampling

In this section, we will introduce recent sampling, then propose forward sampling that implements recent sampling efficiently in $O(1)$ and provide theorectical analysis in Sec. 4.3.

Definition 4.3 (Recent sampling)**.** Recent sampling assigns larger sampling weights to more recent temporal neighbors. The probability a temporal neighbor of node u, e.g., $(v, e_{u,v}^{t'}, t')$ is sampled for learning Z_u^t is proportional to $\exp(c(t' - t))$ where $c \geq 0$ is a constant. This sampling strategy is previously considered in Xu et al. [17] and Wang et al. [16]. Recent sampling has shown to be effective for many scenarios as both recent and some early-day neighbors could provide information. By tuning c, recent sampling may approximate uniform sampling (for a small c) and truncation (for a large c). However, it is hard to implement efficiently via the traditional backward sampling. Not only the existing way to implement recent sampling needs to access all the timestamps of historical interactions to compute those non-uniform sampling probabilities, but also performing non-uniform sampling is rather irregular, which could be much slower than truncation and uniform sampling.

Forward Recent Sampling. For both uniform sampling and recent sampling, if they are achieved through backward sampling as practiced by current methods, the time complexity is at least $O(|N_u^t|)$

as they need to access the entire historical neighbors. We now introduce forward sampling that implements recent sampling in $O(1)$. An overview of the algorithm is demonstrated in Fig. 1 Right.

Intuitively, to abandon the expensive backtracking and sampling procedures, we adopt a dictionary-type table that keeps track of a list of down-sampled temporal neighbors for each node. The storage tables are updated after the model has responded to the previous query but before the next query comes in. So, the update will not introduce inference latency, although the update itself is cheap (of $O(1)$ complexity) and parallelizable on GPUs.

Formally, we denote S_u to be the down-sampled temporal neighbors of node u stored within a GPU hash table. For efficient lookup and update in batches, we allocate fixed-size GPU memory for these tables, where the size s, as a user-specified parameter, is the maximum number of temporal neighbors to be sampled. The current snapshot of table S_u at a timestamp t is denoted as S_u^t. S_u consists of key-value pairs with unique keys. Each key-value pair corresponds to one of the temporal neighbors of u, e.g., $(v, e_{u,v}^{t'}, t') \in \mathcal{N}_u^t$. While we store the edge features $e_{u,v}^{t'}$ as the *values*, the choice of the *keys* is flexible: It can either be the pair of the neighbor ID and timestamp, i.e., (v, t'), or just the neighbor ID v. We name the method using (v, t') as *NLB-edge* and the one using v as *NLB-node*. Different choices of the keys have different sampling properties which we will discuss in Sec. 4.3.

To achieve recent sampling, the tables get updated overtime when new links come in. Specifically, the new temporal neighbor indicated by the new link is inserted into the table through random hashing using the keys. When there is a hash collision, the older temporal neighbor will be replaced by the new one with a probability $\alpha \in (0, 1]$, where α is a hyperparameter. In order for more recent neighbors to be sampled, we set α to be closer to 1.

Example. In Fig. 1, when link (u, v, t_4) arrives, we initialize an update to S_u and S_v, creating new snapshots $S_u^{t_4}$ and $S_v^{t_4}$. Specifically, for NLB-edge which uses tuples of neighbor ID and timestamp as keys, the new temporal neighbor $(v, e_{u,v}^{t_4}, t_4)$ of u will be assigned to position

$$\mathit{hash}(v, t_4) \equiv (q_1 * v + q_2 * t_4) \ (\mathbf{mod}\ s) \tag{2}$$

for fixed large prime numbers q_1 and q_2. For NLB-node which uses neighbor IDs as keys, the temporal neighbor will be assigned to position

$$\mathit{hash}(v) \equiv (q_1 * v) \ (\mathbf{mod}\ s). \tag{3}$$

Let a denote the assigned position for (u, v, t_4). Then, $(v, e_{u,v}^{t_4}, t_4)$ will be inserted into S_u, if there is no hash collision, i.e., $S_u[a]$ being empty. Otherwise, the temporal neighbor will be inserted with a probability α.

This completes the procedure for forward sampling. At future timestamp $t > t_4$, $S_u^{t_4}$ and $S_v^{t_4}$ can be used as the down-sampled temporal neighbors for learning node representations Z_u^t and Z_u^t until new snapshots of S_u and S_v are generated. How NLB encodes these temporal neighbors into node representations will be discussed in Sec. 4.4.

4.3 The properties of forward recent sampling

The sampling properties for different choices of the keys are different. Intuitively, a key with both neighbor IDs and timestamps allows NLB-edge to keep track of more dynamic information about each neighboring node, while hashing with only neighbor IDs allows NLB-node to learn more static information. In the following Theorem 4.4, we show that NLB-edge essentially implements recent sampling (Def. 4.3) with $O(1)$ complexity. The proof can be found in Appendix A. NLB-node actually implements a node-wise recent sampling rather than the standard recent sampling with $O(1)$ complexity, which will be elaborated in Appendix B.

Theorem 4.4 (NLB-edge achieves recent sampling)**.** *Suppose links come in for any node (e.g. u) by following a Poisson point process with a constant intensity (e.g. λ), and suppose a temporal neighbor $(v, e_{u,v}^{t_i}, t_i)$ is inserted into S_u at time t_i, then* $\Pr((v, e_{u,v}^{t_i}, t_i) \in S_u^t) = \exp(\frac{\alpha\lambda}{s}(t_i - t))$.[1]

This result matches the definition of recent sampling (Def. 4.3) where the probability a temporal neighbor gets sampled is proportional to $\exp(c(t_i - t))$. If we instead suppose that each unique neighboring node (e.g. v) of u connects with u following a different Poisson point process (e.g. λ_v), we recover the sampling probabilities of NLB-node.

[1] Poisson point process is a commonly used assumption to model communication networks Lavenberg [45], and is also an assumption used by CAWN [16].

4.4 Node representation generation and NLB prediction

When a prediction query comes, NLB generates the node representations based on the sets of down-sampled temporal neighbors of relevant nodes, as introduced in Sec. 4.2, for such prediction. We first denote the status of node u at time t as r_u^t, which can be viewed as a self feature vector aggregated over time via RNN and will be elaborated later. Node status can be viewed as to collect local information, while the node representation Z_u^t also leverages the status of the temporal neighbors (decided by S_u^t), which can be viewed as to provide more global information. Empirically, we observe that combining both yields the best performance.

Prediction. Given the down-sampled neighbors S_u^t of u, we first retrieve the status of the temporal neighbors $\{r_v^t\}$. We then use attention to aggregate the collected status $\{r_v^t\}$ and the self status r_u^t. The timestamps and edge features for temporal neighbors will also be encoded together. Specifically, let MLP denote a multi-layer perceptron. We adopt

$$Z_u^t = \text{MLP}\left(r_u^t, \sum_{(v,e_{u,v}^{t'},t')\in S_u^t} \alpha_i \text{MLP}(r_v^t, e_{u,v}^{t'}, t')\right), \tag{4}$$

where $\{\alpha_i\} = \text{softmax}(\{w^T\text{MLP}(r_v^t, e_{u,v}^{t'}, t')|(v, e_{u,v}^{t'}, t') \in S_u^t\})$, where w is a learnable vector parameter. A pair of node representations Z_u^t and Z_v^t can be combined and plugged in to classifiers to make link predictions; and a single node representation Z_u^t can be used for node classification. t is encoded via T-encoding (see Def. C.1 in Appendix C) which has been proved to be effective in previous works [16, 17, 46, 47].

Note that a node status r_u^t is updated via $r_u^{t+1} \leftarrow \textbf{RNN}([r_u^t, r_v^t, t, e_{u,v}^t])$ after an event (u, v, t) arrives.

Also, NLB naturally supports efficient aggregation of second-hop temporal neighbors because to get representation for node u, if v is stored in S_u^t, i.e., one of the first-hop temporal neighbors, S_v^t provides the down-sampled second-hop neighbors of u, which can be efficiently accessed. However, in experiment, we use the first hop only as it already provides decent prediction performance. Involving the second hop, albeit bringing more information, will increase the inference latency.

5 Experiments

In this section, we evaluate NLB in its prediction performance and scalability on real-world temporal networks, and further conduct hyperparameter analysis.

5.1 Experimental setup

Datasets. We use six publicly available real-world datasets whose statistics are listed in Appendix Table 6 for experiments. There are datasets with billion-scale temporal links and million-scale nodes. Further details of these datasets can be found in Appendix D. We split the datasets into training, validation and testing data according to the ratio of 70/15/15 while preserving the chronological order.

Downstream tasks and models. We conduct experiments on link prediction with transductive and inductive settings for all datasets. For datasets where node labels are available, we also conduct node classification. For inductive link prediction, we follow previous work and sample the unique nodes in validation and testing data with probability 0.1 and remove them and their associated edges from the networks during the training stage. Following previous work, the temporal graph representation models are trained with self-supervised link prediction task. The models are then directly used to generate node representations for node classification. For fair comparison, the architectures for all downstream models are single-layer perceptrons with ReLU activation. The detailed procedures for inductive evaluation and node classification for NLB are documented in Appendix E.

Baselines. We select four representative baselines, each with two variants. Among the baselines, TGAT [17] proposed aggregating temporal neighborhood information with attention, TGN [15] proposed keeping a memory state for each node that gets update with new interactions, APAN [21] proposed asynchronous graph propagation instead of graph aggregation, and NAT [18] improves link prediction with their joint neighborhood structural features. For TGAT, TGN and APAN, we evaluate their performance and scalability with two sampling strategies: uniform sampling (denoted as *unif*) and truncation of the most recent neighbors (denoted as *trunc*). We evaluate these baselines (excluding NAT) using the implementation in TGL [25] that relies on low-level C++ programming and multi-thread CPU, as it is currently the most efficient framework for them and the original implementation has shown to be significantly slower. For NAT, we adapt it for node classification (named *NAT-node*)

Task	Method	Wikipedia	Reddit	GDELT	MAG	Ubuntu	Wiki-talk
Transductive	TGN-trunc	99.48 ± 0.07	99.65 ± 0.07	98.63 ± 0.05	99.32 ± 0.06	76.09 ± 0.17	84.73 ± 3.57
	TGN-unif	99.44 ± 0.03	99.66 ± 0.05	97.85 ± 0.02	99.24 ± 0.03	78.99 ± 1.69	83.61 ± 0.10
	TGAT-trunc	97.47 ± 0.06	97.84 ± 0.03	98.31 ± 0.02	99.09 ± 0.03	76.71 ± 0.33	81.23 ± 0.06
	TGAT-unif	95.35 ± 0.18	98.15 ± 0.15	97.76 ± 0.01	99.02 ± 0.05	78.59 ± 0.23	80.48 ± 0.01
	APAN-trunc	99.20 ± 0.03	96.46 ± 2.98	97.89 ± 0.16	90.61 ± 1.24	69.90 ± 3.28	82.10 ± 5.67
	APAN-unif	97.90 ± 0.40	97.65 ± 0.20	97.56 ± 0.74	CPU OOM	77.62 ± 2.71	84.32 ± 7.28
	NAT	**99.72 ± 0.03**	**99.90 ± 0.01**	GPU OOM	GPU OOM	**89.65 ± 0.17**	**94.66 ± 0.03**
	NAT-node	99.29 ± 0.07	99.76 ± 0.02	GPU OOM	GPU OOM	87.48 ± 0.49	93.03 ± 0.52
	NLB-edge	99.42 ± 0.08	99.73 ± 0.02	**98.94 ± 0.28**	**99.39 ± 0.02**	87.18 ± 0.55	91.72 ± 0.62
	NLB-node	99.03 ± 0.07	99.67 ± 0.02	**98.80 ± 0.16**	99.15 ± 0.02	87.48 ± 0.64	91.95 ± 0.17
Inductive	TGN-trunc	98.55 ± 0.08	99.44 ± 0.06	98.62 ± 0.01	96.05 ± 0.12	80.70 ± 1.64	87.65 ± 1.01
	TGN-unif	98.33 ± 0.08	99.30 ± 0.01	**98.77 ± 0.08**	96.45 ± 0.26	**85.23 ± 1.66**	87.11 ± 0.30
	TGAT-trunc	96.57 ± 0.04	95.22 ± 0.21	94.36 ± 0.01	98.80 ± 0.01	77.22 ± 0.07	79.97 ± 0.04
	TGAT-unif	93.80 ± 0.15	96.12 ± 0.07	94.05 ± 0.02	98.77 ± 0.01	78.07 ± 0.04	77.03 ± 0.03
	APAN-trunc	96.65 ± 0.07	95.93 ± 2.43	98.48 ± 0.01	CPU OOM	65.19 ± 5.50	78.90 ± 1.11
	APAN-unif	97.23 ± 0.31	96.56 ± 0.32	97.64 ± 0.14	CPU OOM	78.39 ± 2.87	87.86 ± 0.11
	NAT	**99.52 ± 0.03**	**99.78 ± 0.03**	GPU OOM	GPU OOM	84.71 ± 1.01	**92.23 ± 1.16**
	NAT-node	98.47 ± 0.25	99.53 ± 0.21	GPU OOM	GPU OOM	81.51 ± 1.36	79.12 ± 5.14
	NLB-edge	98.43 ± 0.26	99.43 ± 0.07	98.16 ± 0.32	**98.85 ± 0.02**	**86.88 ± 0.62**	90.91 ± 1.14
	NLB-node	98.31 ± 0.30	99.36 ± 0.09	97.14 ± 0.42	**98.79 ± 0.11**	85.47 ± 0.73	**91.22 ± 1.39**

Table 2: Link prediction performance in AUC (mean in percentage ± 95% confidence level). **Bold font** and underline highlight the best performance and the second best performance on average.

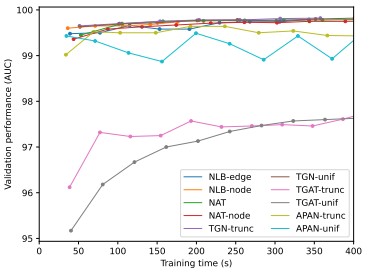

Figure 2: The transductive link prediction validation performance v.s. training time on Reddit. Each dot on the curves gets collected at the end of an epoch.

Method	Wikipedia (AUC)	Reddit (AUC)	GDELT (F1)	MAG (F1)
TGN-trunc	86.85 ± 0.56	63.45 ± 1.21	18.39 ± 1.37	2.33 ± 0.00
TGN-unif	85.99 ± 0.62	**64.38 ± 1.60**	21.38 ± 0.04	2.33 ± 0.00
TGAT-trunc	78.27 ± 1.12	60.54 ± 0.97	22.91 ± 0.03	2.33 ± 0.00
TGAT-unif	84.08 ± 0.41	63.44 ± 2.72	**24.34 ± 0.02**	2.33 ± 0.00
APAN-trunc	86.50 ± 0.60	56.74 ± 0.62	9.14 ± 0.43	3.11 ± 0.00
APAN-unif	87.27 ± 1.02	60.68 ± 1.42	9.90 ± 0.01	CPU OOM
NAT	-	-	-	-
NAT-node	86.65 ± 0.55	60.44 ± 2.36	GPU OOM	GPU OOM
NLB-edge	**89.92 ± 0.20**	62.73 ± 2.27	23.90 ± 0.05	**30.93 ± 0.02**
NLB-node	89.52 ± 0.06	62.93 ± 0.56	23.46 ± 0.08	29.69 ± 0.02

Table 3: Performance of node classification (mean in percentage ± 95% confidence level). **Bold font** and underline highlight the best performance and the second best performance on average.

by removing their joint structural features and directly aggregating their neighborhood representations to generate node representations. Additional details about these baselines can be found in Appendix E.

Hyperparameters. For fair comparison, we fix the maximum number of neighbors to be sampled for each dataset across all methods as specified in Appendix Table 9. When evaluating scalability, we (1) limit the maximum number of CPU threads to be at most 32, (2) use one GPU for all methods and (3) fix the batch sizes across different methods as specified in Appendix Table 10. For the rest of the hyperparameters, if a dataset has been tested previously, we use the set of hyperparameters that are provided by the baseline models. Otherwise, we tune the parameters with grid search and make sure the sizes of different modules such as the node representation and time feature, have dimensions of the same scale for all methods. More detailed hyperparameters are provided in Appendix E. **Hardwares.** We run all experiments using the same Linux device that is equipped with 256 AMD EPYC 7763 64-Core Processor @ 2.44GHz with 2038GiB RAM and one GPU (NVIDIA RTX A6000).

Performance Metrics. For link prediction performance, we evaluation all models with Area Under the ROC curve (AUC), Average Precision (AP) and Mean Reciprocal Rank (MRR) with a large number of negative samples per positive sample (50 for GDELT, and 500 for other datasets, excluding the MAG dataset due to the time and memory constraints). In the main text, the prediction performance in all tables is evaluated in AUC. For node classification, we evaluate with AUC for Wikipedia and Reddit which have two node classes, and we use F1 for GDELT and MAG which have more than two classes. All results are summarized based on 5 time independent experiments with different random seeds and initializations.

Scalability Metrics. To evaluate scalability, we consider both time and energy efficiency. We include (a) average training time (Train), testing time (Test) and inference latency (Inf. Lat.) per epoch in seconds, (b) the total energy consumption from CPUs and GPUs (in Joules) denoted as CPU and GPU. We ensure that there are no other applications running during our evaluations.

		Train	**Test**	**Inf. Lat.**	**CPU (MJ)**	GPU (MJ)		Train	**Test**	**Inf. Lat.**	**CPU (MJ)**	GPU (MJ)		Train	**Test**	**Inf. Lat.**	**CPU (MJ)**	GPU (MJ)
Method																		
TGN-trunc	Wikipedia	13.06	1.65	1.20	2,237	1.04	Ubuntu	15.95	2.05	1.37	3,332	1.75	GDELT	1977	406	302	431,567	174.6
TGN-unif		13.25	1.67	1.23	2,194	1.02		15.55	2.11	1.42	3,326	1.78		4791	1013	777	797,311	389.4
TGAT-trunc		10.68	1.35	0.87	1,882	0.82		12.37	1.70	1.20	3,028	1.30		1123	248	183	306,860	92.7
TGAT-unif		10.70	1.34	0.90	1,779	0.76		13.64	1.93	1.27	3,254	1.37		1493	383	301	251,452	83.8
APAN-trunc		10.67	1.48	0.57	2,034	0.90		12.96	1.94	0.61	2,802	1.36		1530	338	200	485,818	211.2
APAN-unif		10.67	1.51	0.58	1,853	0.84		13.01	2.07	0.62	2,894	1.34		1839	347	187	512,061	188.4
NAT		14.81	1.72	1.11	2,900	1.29		17.52	1.96	1.22	2,853	1.27		-	-	-	-	-
NAT-node		12.28	1.32	0.82	2,621	1.02		15.60	1.60	1.30	2,824	1.23		-	-	-	-	-
NLB-edge		8.28	0.86	0.43	1,609	0.85		11.06	1.30	0.57	2,764	2.24		1323	139	63	252,267	307.6
NLB-node		7.95	0.86	0.43	1,547	0.80		10.74	1.35	0.59	2,748	2.17		1089	118	60	215,820	258.5
Method		Train	**Test**	**Inf. Lat.**	**CPU (MJ)**	GPU (MJ)		Train	**Test**	**Inf. Lat.**	**CPU (MJ)**	GPU (MJ)		Train	**Test**	**Inf. Lat.**	**CPU (GJ)**	GPU (GJ)
TGN-trunc	Reddit	51.24	7.25	4.74	9,218	4.45	Wiki-talk	74.63	11.76	7.61	16,715	9.57	MAG	6845	1266	795	15,041	1.59
TGN-unif		51.97	7.38	4.82	8,853	4.64		85.96	13.77	8.19	18,925	11.2		5704	1224	741	21,334	2.02
TGAT-trunc		38.64	5.46	3.86	7,108	3.24		58.37	8.74	5.88	14,787	5.90		3465	754	518	10,427	1.06
TGAT-unif		40.40	5.64	4.00	7,221	3.25		63.32	9.14	6.35	16,553	6.75		3520	774	541	13,236	1.35
APAN-trunc		41.32	6.42	2.44	7,732	3.43		60.68	10.34	2.98	13,898	6.94		12347	1600	809	22,262	2.45
APAN-unif		41.75	6.55	2.48	7,779	3.47		64.06	10.61	2.88	14,463	7.04		-	-	-	-	-
NAT		54.08	7.43	4.82	11,517	5.59		81.32	10.03	6.08	14,736	7.13		-	-	-	-	-
NAT-node		46.85	6.02	3.63	10,686	4.97		67.28	8.79	4.88	14,492	6.82		-	-	-	-	-
NLB-edge		38.55	3.68	1.88	6,474	3.86		59.18	7.30	3.03	12,277	15.0		3073	593	296	2,684	1.73
NLB-node		34.52	3.68	1.84	6,443	3.70		56.36	7.01	2.99	11,519	13.6		2729	577	279	2,870	1.69

Table 4: Scalability evaluation on all datasets. Note that MJ = 10^6 Joules, and GJ = 10^9 Joules. Note that TGL is adopted to implement baselines including TGN, TGAT, and APAN.

5.2 Results and Discussion

Prediction performance. The node classification performance is given in Table 3 and the link prediction performance in AUC is given in Table 2. The link prediction results in AP and in MRR are given in Appendix Table 7 and Table 8 respectively.

For link prediction, NLB-edge achieves either the best or the second best performances in 10 out of 12 settings (6 datasets with inductive and transductive tasks), while NLB-node closely follows or sometimes marginally surpasses NLB-edge. NAT is the best performing baseline for link prediction because it leverages the joint neighborhood structural features [16, 48] designed for node-pairs. However, NAT cannot be applied to node classification, and it consumes significant GPU memory. We discuss NAT-node in detail later in this section. TGN-trunc and TGN-unif are the second most competitive baselines. They achieve similar performance to NLB-edge and NLB-node sometimes but significantly fall behind on the datasets Ubuntu and Wiki-talk, and 3 of the 4 node classification tasks. One aspect that differentiate Ubuntu and Wiki-talk to the rest of the dataset is that they do not have node or edge features (Table 6). We hypothesize that the benefit of recent sampling is more obvious with the absence of additional features because NLB can gather effective historical information from diverse and recent temporal neighbors. The other baselines usually perform far behind NLB.

For node classification, both NLB-edge and NLB-node achieve similar or better results than the baselines. In MAG, all baselines fail to perform well and we cannot reproduce the scores reported in Zhou et al. [25] (more discussion in Appendix G).

NLB-edge often outperforms NLB-node because NLB-edge tends to encode more temporal neighbors (node IDs plus timestamps) while NLB-node only encodes neighbors with unique node IDs.

Time and energy efficiency. The details of the scalability metrics are shown in Table 4.

NLB-node always have faster training, testing and lower energy consumption than all baselines while NLB-edge closely follows. NLB-node trains 1.63-2.04× faster than NAT and 1.32-4.40× faster than TGN-unif and TGN-trunc.

On inference latency, except for APAN on the Wiki-talk dataset, both NLB-node and NLB-edge shows significant improvements over all settings. APAN aims to improve inference latency but consumes massive CPU memory. On large-scale datasets such as GDELT and MAG, APAN requires significant latency for transfering messages from CPU to GPU and no longer has advantages. Apart from APAN, NLB-node gives 1.63-12.95× speedup on inference latency over all datasets.

Although NLB's GPU energy consumption is slightly larger, it is negligible compared to the CPU energy consumption. Compared to NLB-edge, NAT consumes up to 1.8× more energy while TGN-trunc and TGN-unif consumes 1.2-5.6× and 1.2-7.94× more energy over all datasets respectively. On billion-scale datasets MAG, NLB-edge is 3.88-8.29× more energy efficient than all methods. The large difference between the energy consumed by the CPU v.s. the GPU can be attributed to (1) the default energy costs by CPU when no application is running, and (2) the intensive CPU-GPU communications which is counted toward the CPU cost as it controls the communication (more details in Appendix H).

Over all baselines, truncation is usually more time and energy efficient than uniform sampling. The advantage can be seen more obviously in inference latency. However, since APAN can perform the sampling asynchronously, its inference latency is similar across these two sampling strategies.

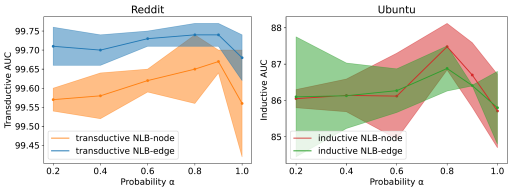

Figure 3: The changes in transductive link prediction test performance on Reddit (Left) and in inductive link prediction test performance on Ubuntu (Right) with respect to α's.

	s	Transductive	Inductive	Node Class.	Train	Test	Inf. Lat.	CPU	GPU
NLB-edge	0	99.59 ± 0.11	98.42 ± 0.17	62.15 ± 0.49	34.26	4.13	1.87	5,699	2.88
	5	99.68 ± 0.03	99.24 ± 0.04	60.53 ± 0.53	39.97	4.90	2.22	6,600	3.57
	10	99.69 ± 0.04	99.44 ± 0.13	60.33 ± 1.32	40.08	4.87	2.21	6,600	3.82
	15	99.65 ± 0.10	99.32 ± 0.04	62.34 ± 1.94	39.49	4.86	2.20	6,576	4.01
	20	99.74 ± 0.03	99.43 ± 0.07	62.73 ± 2.27	39.10	4.87	2.19	6,678	4.10
	25	99.72 ± 0.07	99.47 ± 0.21	60.87 ± 0.14	39.67	4.94	2.24	6,666	4.42
	30	99.67 ± 0.04	99.25 ± 0.09	61.30 ± 0.95	41.32	4.89	2.23	6,930	4.68
NLB-node	0	99.59 ± 0.11	98.42 ± 0.17	62.15 ± 0.49	34.26	4.13	1.87	5,699	2.88
	5	99.60 ± 0.04	99.20 ± 0.02	61.64 ± 4.03	39.03	4.87	2.18	6,460	3.30
	10	99.59 ± 0.04	99.16 ± 0.17	59.16 ± 2.27	39.09	4.85	2.18	6,432	3.56
	15	99.66 ± 0.07	99.23 ± 0.08	61.96 ± 3.15	39.32	4.86	2.22	6,447	3.72
	20	99.67 ± 0.02	99.36 ± 0.09	62.93 ± 0.56	39.09	4.86	2.18	6,541	3.79
	25	99.66 ± 0.07	99.37 ± 0.07	62.30 ± 1.11	39.33	4.91	2.21	6,563	3.88
	30	99.65 ± 0.01	99.38 ± 0.05	66.30 ± 1.62	40.07	4.88	2.19	6,739	4.08

Table 5: The performance and scalability of NLB-edge and NLB-node on the Reddit dataset given different down-sampled neighbor hash table sizes s. $s = 20$ is the setting used for comparison with baselines. CPU and GPU energies are measured in MJ.

We further compare the efficiency of different sampling approaches in Appendix I, which shows that forward sampling is significantly faster. We also plot the model convergence curves in Fig. 2 on the link prediction validation performance v.s. CPU/GPU wall-clock training time on Reddit. NLB-edge and NLB-node are among the fastest to converge to a high performance.

Comparisons between NLB and NAT-node. Overall, NAT-node shows improvement over NAT on all of the scalability metrics. However, it does not outperform NLB in either computational efficiency or prediction performance. As shown by Table 4, over all datasets, NAT-node is still slower than NLB-edge (1.14-1.48× slower in training and 1.61-1.91× slower in inference). It still causes GPU out-of-memory error for large-scale datasets (GDELT and MAG). We think the computational inefficiency may be caused by the storage and maintenance of their neighborhood representations which are primarily used for structural features construction. Furthermore, NAT-node does not perform as well as NLB-edge on link prediction (Table 2) and node classification (Table 3).

5.3 Additional analysis

The effect of different α's and the optimality of recent sampling. We study how changes in the replacement probability α affects the link prediction performance. We show the transductive task on Reddit in Fig. 3 Left (with inductive task in Appendix Fig. 4) and the inductive task on Ubuntu in Fig. 3 Right (with transductive task in Appendix Fig. 5). We observe a general trend for both NLB-edge and NLB-node that increasing α from 0.2 to 0.8 or 0.9 improves the performance. However, when $\alpha = 1$, the performance drops significantly. When $\alpha = 1$, new interactions will always replace the existing down-sampled temporal neighbors, which is similar to the strategy of truncation. This demonstrates the sub-optimality of truncation and the necessity of randomness in neighbor sampling. For smaller α, less recent neighboring nodes will be kept in the hash table, which performs more like uniform sampling. The performance decay for small α indicates the sub-optimality of uniform sampling. Therefore, recent sampling is important for prediction performance.

The effect of the hash table size s. In Table 5, we show how changes in s affect the performance and scalability of NLB-edge and NLB-node respectively using the Reddit Dataset. While $s = 0$ is usually the worst-performing in both transductive and inductive link prediction, it has similar performance in node classification and it is more time and energy-efficient as expected. In contrast, when $s = 5, 10$, the performance on node classification achieves the worst for both NLB-edge and NLB-node. Then, both performance and scalability metrics have an increasing trend as s gets larger. However, the performance for NLB-node is still increasing at $s = 30$, while NLB-edge shows decreases. The link prediction performance of NLB-edge is better than NLB-node but the node classification performance of NLB-node is better. We hypothesize that NLB-edge can more easily overfit data as s becomes large. Finally, NLB-node is always slightly more time and energy-efficient than NLB-edge.

We give the evaluation of MRR for various α's and s' in Appendix Table 13 and Table 14 respectively.

6 Limitations and Future Work

We are aware of the limitation of NLB and highlight some future research directions. For industry-level graphs, it may not be possible to maintain the down-sampled temporal neighbors of all nodes in a single GPU. In the future, we plan to adopt distributed GPUs by partitioning the down-sampled neighbors of different nodes into different GPUs. We will guarantee that the memory used for each node is contained within the same GPU so that the maintenance can still be fast.

7 Conclusion

In this work, we introduced NLB which abandons the traditional time-consuming temporal neighbor backtracking and sampling while adopting the newly proposed forward recent sampling. We proved that NLB-edge achieves recent sampling of links and NLB-node achieves recent sampling of nodes. Our extensive experiments demonstrate that NLB is effective in both prediction performance and scalability while significantly improving inference latency and energy efficiency.

8 Acknowledgement

The authors would like to thank Hongkuan Zhou of USC for the time to discuss the code for TGL [25]. We would also like to thank all reviewers for providing valuable feedback for improving this work. Y. Luo and P. Li are supported by NSF awards PHY-2117997, IIS-2239565.

References

[1] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015. 1

[2] Andrew Z Wang, Rex Ying, Pan Li, Nikhil Rao, Karthik Subbian, and Jure Leskovec. Bipartite dynamic representations for abuse detection. In *KDD*, pages 3638–3648, 2021.

[3] Pan Li, Yen-Yu Chang, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. F-fade: Frequency factorization for anomaly detection in edge streams. In *WSDM*, 2021. 1

[4] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7), 2007. 1

[5] Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, pages 447–456, 2009. 1

[6] Smriti Bhagat, Graham Cormode, and S. Muthukrishnan. Node classification in social networks. *Social Network Data Analytics (Springer)*, 2011. 1

[7] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, Pascal Poupart, and Karsten M. Borgwardt. Representation learning for dynamic graphs: A survey. *JMLR*, 21:70:1–70:73, 2019. URL https://api.semanticscholar.org/CorpusID:216608194.

[8] Meng Liu, Yue Liu, KE LIANG, Wenxuan Tu, Siwei Wang, sihang zhou, and Xinwang Liu. Deep temporal graph clustering. In *ICLR*, 2024. 1

[9] Xuhong Wang, Baihong Jin, Ying Du, Ping Cui, and Yupu Yang. One-class graph neural networks for anomaly detection in attributed networks. In *Neural Comput & Applic*, 2021. 1

[10] Jundong Li, Harsh Dani, Xia Hu, and Huan Liu. Radar: Residual analysis for anomaly detection in attributed networks. In *IJCAI*, page 2152–2158, 2017.

[11] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, page 594–602, 2019.

[12] Xu Yuan, Na Zhou, Shuo Yu, Huafei Huang, Zhikui Chen, and Feng Xia. Higher-order structure based anomaly detection on attributed networks. In *2021 IEEE International Conference on Big Data (Big Data)*, page 2691–2700, 2021. 1

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 1

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017. 1, 2

[15] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML 2020 Workshop on GRL*, 2020. 1, 2, 4, 6, 14, 16, 17

[16] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *ICLR*, 2021. 2, 4, 5, 6, 8

[17] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *ICLR*, 2020. 2, 4, 6, 14, 16, 17

[18] Yuhong Luo and Pan Li. Neighborhood-aware scalable temporal network representation learning. In *LoG*, 2022. 2, 3, 6, 17

[19] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*, 2019. 2, 14, 15

[20] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *ICLR*, 2019. 2

[21] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, and Bowen Sun. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2628–2638, 2021. 1, 2, 3, 4, 6, 16

[22] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018. 2

[23] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *NeurIPS*, page 4563–4572, 2018.

[24] Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. DSpar: An embarrassingly simple strategy for efficient GNN training and inference via degree-based sparsification. *TMLR*, 2023. 2

[25] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. Tgl: A general framework for temporal gnn training on billion-scale graphs. In *Proceedings of the VLDB Endowment*, 2022. 2, 3, 4, 6, 8, 10, 15, 16, 17

[26] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, 2018. 2

[27] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. Dynamic network embedding: An extended approach for skip-gram based network embedding. In *IJCAI*, 2018.

[28] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. dynnode2vec: Scalable dynamic network embedding. In *International Conference on Big Data (Big Data)*. IEEE, 2018.

[29] Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. In *IJCAI*, 2019.

[30] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW*, 2018.

[31] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM*, 2020. 2

[32] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: deep temporal reasoning for dynamic knowledge graphs. In *ICML*, 2017. 2

[33] Amauri Souza, Diego Mesquita, Samuel Kaski, and Vikas Garg. Provably expressive temporal graph networks. In *NeurIPS*, 2022.

[34] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction. In *NeuRIPS*, 2022.

[35] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks? In *ICLR*, 2023. 2

[36] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. In *NeuRIPS*, 2023. 2

[37] Maciej Besta, Afonso Claudino Catarino, Lukas Gianinazzi, Nils Blach, Piotr Nyczyk, Hubert Niewiadomski, and Torsten Hoefler. HOT: Higher-order dynamic graph representation learning with efficient transformers. In *The Second Learning on Graphs Conference*, 2023.

[38] Yuxing Tian, Yiyan Qi, and Fan Guo. Freedyg: Frequency enhanced continuous-time dynamic graph model for link prediction. In *ICLR*, 2024. 2

[39] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old MLPs new tricks via distillation. In *ICLR*, 2022. 2

[40] Zhichun Guo, William Shiao, Shichang Zhang, Yozen Liu, Nitesh V. Chawla, Neil Shah, and Tong Zhao. Linkless link prediction via relational distillation. In *ICML*, 2023.

[41] Yiwei Wang, Bryan Hooi, Yozen Liu, and Neil Shah. Graph explicit neural networks: Explicitly encoding graphs for efficient and accurate inference. In *WSDM*, 2023. 2

[42] Hongkuan Zhou, Ajitesh Srivastava, Hanqing Zeng, Rajgopal Kannan, and Viktor Prasanna. Accelerating large scale real-time gnn inference using channel pruning. In *VLDB*, 2021. 2

[43] Hongkuan Zhou, Da Zheng, Xiang Song, George Karypis, and Viktor Prasanna. Disttgl: Distributed memory-based temporal graph neural network training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, 2023. 3

[44] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael M. Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. 3, 17

[45] Stephen Lavenberg. *Computer performance modeling handbook*. Elsevier, 1983. 5

[46] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Self-attention with functional time representation learning. In *NeurIPS*, 2019. 6, 14

[47] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019. 6, 14

[48] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *NeurIPS*, 2020. 8

[49] Yunyu Liu, Jianzhu Ma, and Pan Li. Neural predicting higher-order patterns in temporal networks. In *WWW*, 2022. 14

[50] Wikipedia edit history dump. URL `https://meta.wikimedia.org/wiki/Data_dumps`. 14

[51] Reddit data dump. URL `http://files.pushshift.io/reddit/`. 15

[52] Kalev Leetaru and Philip A. Schrodt. Gdelt: Global data on events, location, and tone. *ISA Annual Convention*, 2013. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.686.6605`. 15

[53] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021. 16

[54] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. 16

[55] Stack exchange data dump. URL `https://archive.org/details/stackexchange`. 16

[56] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *WSDM*, 2017. 16

[57] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Governance in social media: A case study of the wikipedia promotion process. In *International AAAI Conference on Weblogs and Social Media*, 20q0. 16

[58] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018. 16

[59] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 17

A The proof of Theorem 4.4 (NLB-edge achieves recent sampling)

Theorem A.1 (NLB-edge achieves recent sampling (Theorem 4.4 Restated))**.** *Suppose links come in for any node (e.g. u) by following a Poisson point process with a constant intensity (e.g. λ), and suppose a temporal neighbor $(v, e_{u,v}^{t_i}, t_i)$ is inserted into S_u at time t_i, then* $\Pr((v, e_{u,v}^{t_i}, t_i) \in S_u^t) = \exp(\frac{\alpha\lambda}{s}(t_i - t))$.

Proof. Let $N_i \coloneqq (v_i, e_{u,v_i}^{t_i}, t_i)$ and $N_i \in \mathcal{N}_u^t$ be one of the historical temporal neighbors of u, where $i \in \left[1, |\mathcal{N}_u^t|\right]$. By construction, N_i is hashed to the position $\mathit{hash}(v_i, t_i)$ (Eq. 2). The probability that any other temporal neighbors is inserted to S_u is α as we define in Sec. 4.2. Since we suppose N_i is already inserted to S_u, we only need to evaluate the probability that it does not get replaced by another temporal neighbor. We know from the Poisson process that

$$\mathbb{E}\Big[\#\text{ links of } u \text{ arrive since } t_i\Big] = \lambda(t - t_i). \tag{5}$$

On average, each link has probability $\frac{\alpha}{s}$ of replacing N_i. Then,

$$\mathbb{E}\Big[\#\text{ links replace } S_u[\mathit{hash}(v_i, t_i)] \text{ since } t_i\Big] = \frac{\alpha\lambda(t - t_i)}{s}. \tag{6}$$

Thus, the intensity of links being inserted to the same position as N_i is $\frac{\alpha\lambda}{s}$.

By the property of a Poisson process, the probability that none of the links get inserted to the same position as N_i from t_i to t is $\exp(\frac{\alpha\lambda}{s}(t_i - t))$. Overall,

$$\Pr(N_i \in S_u^t) = \exp(\frac{\alpha\lambda}{S}(t_i - t)), \tag{7}$$

completing the proof.

We have shown that $\Pr((v, e_{u,v}^{t_i}, t_i) \in S_u^t)$ is proportional to $\exp(c(t_i - t))$ for $c \geq 0$. Thus, NLB-edge achieves recent sampling.

B The sample properties of NLB-node

In this section, we first propose a new sampling strategy called *recent node-wise sampling*, we then show that NLB-node essentially achieves recent node-wise sampling in $O(1)$.

Recent node-wise sampling. Instead of sampling temporal neighbors, recent node-wise sampling considers sampling unique neighboring nodes: Given a center node u and its neighboring node v, only the latest interactions till time t between u and v can be sampled and the down-sampled neighbor nodes do not contain duplicates. This strategy can speed up the aggregation of a node's neighborhood as it reduces the number of sampled candidates.

We now prove that NLB-node achieves recent node-wise sampling.

Theorem B.1 (NLB-node achieves recent node-wise sampling)**.** *Suppose each unique node (e.g. $v_i \in \mathcal{V}$) interacts with a center node (e.g. u) by following a different Poisson point process with a constant intensity (e.g. λ_{v_i}), and suppose the latest interaction between a center node and a neighboring node, (e.g. $(v_i, e_{u,v_i}^{t_i}, t_i)$) is inserted into S_u at time t_i, then* $\Pr((v_i, e_{u,v_i}^{t_i}, t_i) \in S_u^t) = \prod_{v_j \in \mathcal{V}\setminus\{v_i\}} \Big(\frac{s-1}{s} + \frac{1}{s}\exp(\alpha\lambda_{v_j}(t_i - t))\Big)$.

Proof. The probability that any other temporal neighbors is inserted to S_u is α as we define in Sec. 4.2. Similar to the argument in Theorem 4.4, since we suppose the latest link with between node v_i and u, i.e., $(v_i, e_{u,v_i}^{t_i}, t_i)$, is already inserted to S_u, we only need to evaluate the probability that it does not get replaced by another node in $\mathcal{V}$. NLB-node computes the hash values based on neighbor node IDs without timestamps. Thus, the probability a different node v_j has the same hash value as v_i is the following,

$$\Pr(\mathit{hash}(v_j) = \mathit{hash}(v_i)) = \frac{1}{s}. \tag{8}$$

Measurement	Wikipedia	Reddit	GDELT	MAG	Ubuntu	Wiki-talk
nodes	9K	11K	17K	122M	159K	1.1M
temporal links	157K	672K	191M	1.3B	964K	7.8M
node classes	2	2	81	152	0	0
labels	217	366	42M	1.4M	0	0
node features	0	0	413	768	0	0
edge features	172	172	186	0	0	0

Table 6: Summary of dataset statistics.

We also have

$$\Pr(\textit{hash}(v_j) \neq \textit{hash}(v_i)) = \frac{s-1}{s}. \tag{9}$$

For an arbitrary $v_j \in \mathcal{V}$ where $\textit{hash}(v_i) = \textit{hash}(v_j)$, during (t_i, t),

$$\mathbb{E}\Big[\#\text{ links between } u \text{ and } v_j \text{ arrive since } t_i\Big] = \lambda_{v_j}(t - t_i). \tag{10}$$

Notice that if v_j never interacts with u, then $\lambda_{v_j} = 0$. Suppose there are interactions between u and v_j, then each link between u and v_j has probability α to collide with $(v_i, e^{t_i}_{u,v_i}, t_i)$ stored in S^t_u. Then,

$$\mathbb{E}\Big[\#\text{ links between } u \text{ and } v_j \text{ replace } S_u[\textit{hash}(v_i)] \text{ since } t_i\Big] = \alpha\lambda_{v_j}(t - t_i). \tag{11}$$

Thus, the intensity of links between u and v_j being inserted to $S_u[\textit{hash}(v_i)]$ is $\alpha\lambda_{v_j}$. The probability none of the links associated with node v_j is inserted during (t_i, t) is then $\exp(\alpha\lambda_{v_j}(t_i - t))$, according to the property of a Poisson process.

For the event $(v_i, e^{t_i}_{u,v_i}, t_i) \in S^t_u$ to happen, it has to be true that for all node v_j that $v_j \neq v_i$, either $\text{hash}(v_j) \neq \text{hash}(v_i)$ or none of the links associated with node v_j is inserted to S_u during (t_i, t). Overall,

$$\Pr((v_i, e^{t_i}_{u,v_i}, t_i) \in S^t_u) = \prod_{v_j \in\, \mathcal{V}\setminus\{v_i\}} \left(\frac{s-1}{s} + \frac{1}{s}\exp(\alpha\lambda_{v_j}(t_i - t))\right), \tag{12}$$

completing the proof.

This achieves recent node-wise sampling because as the difference between the current timestamp t and the timestamp t_i of the lastest interaction with a unique neighboring node gets larger, $\Pr((v_i, e^{t_i}_{u,v_i}, t_i) \in S^t_u)$ decays exponentially. The decay factor is controllable by α, which is similar to the constant c in recent sampling of edges (Def. 4.3). As α approaches 0, the probability for each node of being sampled is approaching a constant regardless of recency, which is similar to uniform sampling. As α approaches 1, as long as there is collision, older nodes in S_u will always be replaced by more recent nodes, which is similar to truncation.

C T-encoding

Our temporal features are encoded via a T-encoding technique with the definition below.

Definition C.1 (T-encoding)**.** For any input timestamps t_i, we adopt Fourier features to encode them before using them as features, i.e., with learnable parameter ω_i's, $1 \leq i \leq d$,

$$\text{T-encoding}(t) = [\cos(\omega_1 t), \sin(\omega_1 t), ..., \cos(\omega_d t), \sin(\omega_d t)]. \tag{13}$$

It has shown to be effective for TGRL [15–17, 46, 47, 49].

D Dataset Description

The following are the detailed descriptions of the six datasets we tested.

[leftmargin=*]Wikipedia[2] [19, 50] records edit events on wiki pages. It is a bipartite graph where a set of nodes represents the editors and another set represents the wiki pages. The stamped link

[2] http://snap.stanford.edu/jodie/wikipedia.csv

Task	Method	Wikipedia	Reddit	GDELT	MAG	Ubuntu	Wiki-talk
Transductive	TGN-trunc	99.44 ± 0.08	99.59 ± 0.09	98.34 ± 0.10	99.30 ± 0.06	81.61 ± 0.20	87.03 ± 2.47
	TGN-unif	99.36 ± 0.04	99.60 ± 0.05	96.96 ± 0.05	99.21 ± 0.04	83.27 ± 0.96	87.24 ± 0.16
	TGAT-trunc	97.65 ± 0.04	97.98 ± 0.03	98.12 ± 0.01	99.08 ± 0.03	80.62 ± 0.39	84.81 ± 0.06
	TGAT-unif	94.61 ± 0.23	98.12 ± 0.15	97.54 ± 0.01	99.01 ± 0.05	81.82 ± 0.12	84.94 ± 0.01
	APAN-trunc	99.04 ± 0.03	95.73 ± 2.96	96.79 ± 0.32	87.85 ± 1.68	72.20 ± 3.94	84.13 ± 4.44
	APAN-unif	97.14 ± 0.67	96.57 ± 0.24	96.21 ± 1.35	CPU OOM	78.61 ± 1.61	85.68 ± 6.00
	NAT	**99.72 ± 0.02**	**99.89 ± 0.01**	GPU OOM	GPU OOM	**90.88 ± 0.19**	**95.18 ± 0.03**
	NAT-node	99.13 ± 0.12	99.72 ± 0.02	GPU OOM	GPU OOM	87.75 ± 0.55	93.34 ± 0.55
	NLB-edge	99.30 ± 0.11	99.69 ± 0.02	**98.79 ± 0.39**	**99.38 ± 0.02**	87.52 ± 0.48	92.05 ± 0.62
	NLB-node	98.77 ± 0.09	99.60 ± 0.04	**98.62 ± 0.19**	99.10 ± 0.02	87.61 ± 0.69	92.11 ± 0.08
Inductive	TGN-trunc	98.42 ± 0.08	99.38 ± 0.07	**98.38 ± 0.04**	96.99 ± 0.08	82.27 ± 1.17	88.46 ± 0.64
	TGN-unif	97.99 ± 0.13	99.21 ± 0.03	**98.48 ± 0.15**	97.21 ± 0.14	**85.70 ± 1.58**	88.10 ± 0.13
	TGAT-trunc	96.75 ± 0.02	95.77 ± 0.21	94.64 ± 0.01	98.78 ± 0.01	81.53 ± 0.14	84.10 ± 0.05
	TGAT-unif	92.94 ± 0.15	96.33 ± 0.09	93.32 ± 0.02	98.75 ± 0.01	82.17 ± 0.08	82.21 ± 0.05
	APAN-trunc	95.18 ± 0.91	96.02 ± 2.06	97.99 ± 0.03	CPU OOM	68.60 ± 2.41	80.47 ± 0.35
	APAN-unif	95.73 ± 0.77	96.03 ± 1.31	96.55 ± 0.31	CPU OOM	78.64 ± 2.66	88.82 ± 0.28
	NAT	**99.48 ± 0.02**	**99.76 ± 0.03**	GPU OOM	GPU OOM	**86.69 ± 1.03**	**93.27 ± 0.88**
	NAT-node	97.78 ± 0.42	99.36 ± 0.44	GPU OOM	GPU OOM	82.68 ± 0.88	84.12 ± 2.75
	NLB-edge	98.20 ± 0.32	99.30 ± 0.07	97.95 ± 0.41	**98.86 ± 0.03**	**86.16 ± 0.76**	90.54 ± 1.05
	NLB-node	98.00 ± 0.47	99.18 ± 0.10	96.81 ± 0.56	**98.78 ± 0.13**	84.41 ± 0.83	90.82 ± 1.40

Table 7: Link prediction performance in average precision (AP) (mean in percentage ± 95% confidence level). **Bold font** and underline highlight the best performance and the second best performance on average.

Task	Method	Wikipedia	Reddit	GDELT	Ubuntu	Wiki-talk
Transductive	TGN-trunc	60.28 ± 4.09	74.38 ± 1.72	75.78 ± 8.67	20.94 ± 6.72	31.00 ± 5.02
	TGN-unif	52.05 ± 4.65	65.36 ± 4.49	71.78 ± 0.16	21.87 ± 3.60	22.72 ± 6.55
	TGAT-trunc	45.91 ± 1.15	60.19 ± 1.05	79.11 ± 0.04	22.37 ± 0.97	30.01 ± 0.84
	TGAT-unif	23.17 ± 0.46	45.90 ± 0.47	74.89 ± 0.05	16.61 ± 0.44	18.01 ± 0.43
	APAN-trunc	26.51 ± 1.46	43.68 ± 5.49	74.49 ± 4.14	5.79 ± 1.81	18.91 ± 9.17
	APAN-unif	30.59 ± 3.51	36.29 ± 10.12	72.53 ± 0.22	11.60 ± 4.22	9.99 ± 1.77
	NAT	**77.51 ± 22.83**	**90.40 ± 4.63**	GPU OOM	**46.38 ± 9.14**	**74.17 ± 28.67**
	NAT-node	39.58 ± 6.21	39.89 ± 8.27	GPU OOM	18.93 ± 3.10	21.32 ± 3.59
	NLB-edge	61.90 ± 2.31	75.40 ± 1.85	**84.45 ± 0.84**	24.34 ± 1.89	32.57 ± 1.49
	NLB-node	54.42 ± 3.08	67.65 ± 5.31	76.60 ± 8.43	25.43 ± 0.97	31.95 ± 2.64
Inductive	TGN-trunc	54.29 ± 3.13	58.20 ± 7.35	74.37 ± 1.26	19.58 ± 2.99	27.93 ± 2.94
	TGN-unif	44.26 ± 2.32	53.02 ± 2.04	**81.69 ± 3.73**	18.14 ± 9.87	25.84 ± 0.68
	TGAT-trunc	45.18 ± 1.56	50.67 ± 5.39	59.90 ± 0.59	22.41 ± 0.58	26.15 ± 1.50
	TGAT-unif	15.60 ± 1.39	21.14 ± 6.09	51.72 ± 1.06	12.02 ± 1.03	20.51 ± 1.01
	APAN-trunc	18.92 ± 0.28	34.85 ± 4.38	77.60 ± 0.23	7.16 ± 1.01	15.56 ± 3.52
	APAN-unif	19.83 ± 1.77	38.15 ± 4.45	49.87 ± 3.95	11.63 ± 9.75	17.46 ± 5.45
	NAT	**75.02 ± 22.01**	**89.53 ± 4.08**	GPU OOM	**37.81 ± 12.40**	**39.02 ± 24.29**
	NAT-node	34.56 ± 4.18	50.58 ± 9.29	GPU OOM	19.64 ± 4.43	19.60 ± 6.73
	NLB-edge	49.94 ± 2.12	61.74 ± 1.85	**81.60 ± 4.17**	21.13 ± 2.44	24.26 ± 4.60
	NLB-node	44.59 ± 6.36	53.43 ± 8.03	71.58 ± 14.65	21.46 ± 2.97	25.39 ± 1.87

Table 8: Link prediction performance in Mean Reciprocal Rank (MRR) with large number of negative samples per positive sample (mean in percentage ± 95% confidence level). For the largest-scale datasets GDELT and MAG, we use 50 negative samples while for other datasets we use 500. MAG is not evaluated because it takes a significantly long time to run all models with a large number of negative samples. **Bold font** and underline highlight the best performance and the second best performance on average.

represents the edit events. The edge features are extracted from the contents of wiki pages. The node labels are binary which indicates whether a user is banned from posting. The original data dump is under the Creative Commons Attribution-Share-Alike 3.0 License and the Terms of Use is on the website (https://dumps.wikimedia.org/legal.html). The derived dataset by Kumar et al. [19] is open-sourced here. Reddit[3] [19, 51] is a dataset of the post events by users on subreddits. It is also an attributed bipartite graph between users and subreddits. The node labels are also binary values that indicate whether a user is banned from posting to a subreddit. The original dataset has the Terms of Use in the website (https://pushshift.io/signup). The derived dataset by Kumar et al. [19] is open-sourced here. GDELT[4] [25] is a Temporal Knowledge Graph (TKG) originated from the GDELT Event Database [52] and adapted by TGL for richer node and edge features. It is a large-scale dataset with more than 100M links. It records events from news and articles in over 100 languages every 15 minutes. The nodes represents actors and the edges represent events that happen between a pair of actors. The node features represent the CAMEO codes of the actors and the link features represent the CAMEO codes of the events. The labels are the countries where the actors were located when the events happen. According to the Terms of Use in the website (https://www.gdeltproject.org/), it is an open platform for research and analysis of global society and thus all datasets released by the GDELT Project are available for unlimited and unrestricted use for any academic, commercial, or governmental use of any kind without fee. The derived dataset by Zhou et al. [25] is

[3] http://snap.stanford.edu/jodie/reddit.csv

[4] https://github.com/tedzhouhk/TGL

Params	Wikipedia	Reddit	GDELT	MAG	Ubuntu	Wiki-talk
s	20	20	10	2	20	20

Table 9: Number of sampled temporal neighbors for each dataset for all methods. GDELT and MAG use smaller s because of CPU and GPU size limits.

Params	Wikipedia	Reddit	GDELT	MAG	Ubuntu	Wiki-talk
Batch size	100	100	5000	5000	600	1000

Table 10: The batch sizes used for scalability evaluation for each dataset for all methods.

open-sourced here. MAG[5] Zhou et al. [25] is a homogeneous sub-graph of the heterogeneous MAG240M graph in OGB-LSC [53] extracted by TGL. It is another large-scale dataset with more than 100 million nodes and 1.3 billion links. It is a paper-paper citation network where each node in MAG represents one academic paper. The temporal edges represent citation of one paper to another with timestamp representing the year when the paper is published. The node features are the embeddings of the abstract of the paper generated with RoBERTa [54]. The node labels are the arXiv subject areas. The original dataset is licensed ODC-BY License. The derived dataset by Zhou et al. [25] is open-sourced here. Ubuntu[6] [55, 56] is a dataset that records the events on a stack exchange web site called Ask Ubuntu.[7] The nodes represent users and there are three different types of edges, (1) user u answering user v's question, (2) user u commenting on user v's question, and (3) user w commenting on user u's answer. The original data dump is cc-by-sa 4.0 licensed. The derived dataset by Paranjape et al. [56] is open-sourced here. Wiki-talk[8] [56, 57] is a dataset that represents the edit events on Wikipedia user talk pages. The dataset spans approximately 5 years so it accumulates a large number of nodes and edges. This is a large-scale dataset with more than 1M nodes. The original data dump is under the Creative Commons Attribution-Share-Alike 3.0 License. The derived dataset by Paranjape et al. [56] is open-sourced here.

E Baselines, hyperparameters and the experiment setup

TGAT [17] adapts GAT [58] for static graphs to dynamic graphs. It aggregates messages from dynamic neighbors via an attention mechanism. While it proposed recent sampling, its implementation is significantly inefficient because it has to calculate the recency weights online before sampling. Thus, we only consider uniform sampling and truncation. We use 2 attention heads and 100 hidden dimensions. We only consider aggregation from the first-hop neighbors because aggregating second-hop neighbors introduces even more severe latency for sampling. We also do not observe significant improvement in prediction performance using second-hop neighbors. We do not find an official licensed original implementation for this work and we adopt the implementation by TGL [25].

TGN [15] is a recently proposed SOTA temporal graph representation learning approach. It keeps track of a memory state for each node and update with new interactions. We train TGN with 100 dimensions for each of memory module, time feature and node embedding. We also only consider sampling the first-hop neighbors because of computational efficiency. Their source code[9] is licensed under the Apache-2.0 License but we adopt the implementation by TGL.

APAN [21] is a recent approach designed for low inference latency. It prepares the neighborhood information forward with graph propagation instead of aggregation. While it inspires our approach, it does not support recent sampling and has shown to consume significant CPU memory while performing subpar to SOTA methods. The reason is it stores the messages for each node in CPU which consumes massive CPU memory. We train APAN with 100 dimensions for the node embeddings and 10 dimensions for the mailbox. Their original implementation[10] is licensed under the MIT License but we adopt the implementation by TGL.

[5] https://github.com/tedzhouhk/TGL
[6] https://snap.stanford.edu/data/sx-askubuntu.html
[7] http://askubuntu.com/
[8] https://snap.stanford.edu/data/wiki-talk-temporal.html
[9] https://github.com/twitter-research/tgn
[10] https://github.com/WangXuhongCN/APAN

TGL [25] is a very recent framework that support efficient and scalable temporal graph representation learning. It proposes temporal neighbor sampling with multi-core CPUs in parallel. Overall, it is shown to achieve on average 13× speedup for training and 173× speedup on neighbor sampling. For all of the baselines above, we evaluate them using the implementation by TGL. For the GDELT and the MAG datasets, since the node and edge features do not fit in our GPU, we set the `all_on_gpu` flag to `false`. For all other datasets, we set it to `true`. This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. The source code licensed under the Apache-2.0 License is provided here.[11]

NAT [18] is also a very recent work that achieves SOTA link prediction performance for dynamic graphs. It proposed dictionary-type node representations to replace the traditional single-vector node representations which keep track of historical neighborhood information and avoid the expensive backward neighbor sampling. However, it relies on the construction of joint neighborhood features which only work with node pairs and cannot be easily generalized to node classification. Therefore, we evaluate it on link prediction only. We adapted NAT for node classification (called **NAT-node**) by removing the joint structural features and only aggregating their neighborhood representations to generate node representations. We also only use the first-hop neighborhood. Other than that, we follow the hyperparameter settings documented in their paper. Their implementation[12] is licensed under the MIT License.

NLB (our work). For both NLB-node and NLB-edge, we tune the node representation and the node status dimensions between 50 to 100. We use two attention heads and we tune α and dropout between 0 to 1 with grid search.

Lastly, we note that the Adam optimizer [59] is used for all baselines.

E.1 Inductive evaluation of NLB

Our evaluation pipeline for inductive learning is similar to NAT [18] and different from other baselines. For backward sampling methods such as TGN [15] and TGAT [17], when they do inductive evaluations, they can access the entire training and evaluation data for aggregation, including events that are masked for inductive test. However, NLB's forward sampling prepares the down-sampled neighbors forward for future use and ignores the masked events during the training phase. Thus, by the end of the training, even all historical events become accessible, NLB cannot leverage them. Therefore, to ensure a fair comparison, after training, NLB processes the **full** train and validation data with all nodes unmasked to gather the down-sampled temporal neighbors from the complete set, and then processes the test data. Note that in this last pass over the **full** train and validation data, we only perform the forward sampling and do not perform training anymore.

E.2 Node classification preprocessing

For node classification, we pre-process and combine the node labels to links so that each link can have a source label, a target label or both. For each label y_u^t, we assign the label to the first link of u that appears since time t. If u is a source node, the label is assigned to source label and vice versa. While processing the link stream, if a link has source or target labels, we record the node representations of the source or target node at that time and generate a prediction for the node classes.

F Evaluation on TGB

TGB [44] is adopted by a lot of TGRL methods for standardizing the evaluation. Our main evaluation does not use TGB because it is insufficient for computational evaluation which involves billion-scale graphs. We provide some additional experiments for NLB using TGB here in Table 11. It verifies that NLB can match or outperform TGN in link prediction.

G TGL Node Classification for MAG

We fail to reproduce the node classification scores for MAG over all of the baselines implemented in TGL [25]. Our hypothesis is as follows. TGL has reported the scores for MAG based on an

[11] https://github.com/tedzhouhk/TGL

[12] https://github.com/Graph-COM/Neighborhood-Aware-Temporal-Network

Method	tgbl-review	tgbl-coin	tgbl-comment
TGN	34.90 ± 2.00	58.60 ± 3.70	37.90 ± 2.10
DyRep	22.00 ± 3.00	45.20 ± 4.60	28.90 ± 3.30
EdgeBank(tw)	2.50	58.00	14.90
NLB-edge	35.08 ± 2.20	61.92 ± 1.43	38.84 ± 2.17
NLB-node	37.13 ± 0.40	58.66 ± 2.14	36.91 ± 0.71

Table 11: Link prediction performance in Mean Reciprocal Rank (MRR) evaluated on TGB.

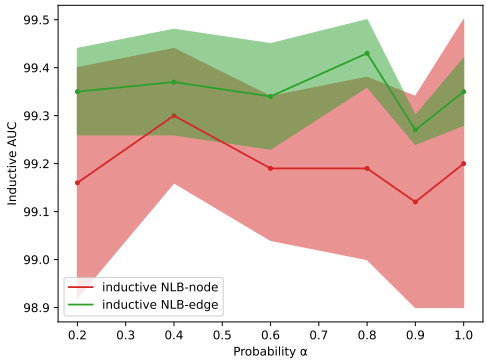

Figure 4: The changes in inductive link prediction test performance with respect to different α on the Reddit dataset.

implementation with an issue. For the backtracking and sampling of temporal neighbors to learn representation at time t, TGL considers temporal neighbors of the same timestamp t which may cause information leakage. We noticed this issue in their multiple-gpu training script. We evaluate TGL using their single-gpu training script which does not have the issue and results in the worse prediction for MAG as shown in Table 3.

H The large difference between CPU and GPU energy consumption

While we directly obtained the numbers from the PyJoules[13] output, we observe a large difference between CPU and GPU energy consumption. We find that it can be attributed to the following factors:

(1) Both the CPU and GPU have default energy costs even when no applications are running, as there may be background applications that constantly require energy. Within 1 minute, the default energy cost for the CPU is roughly 9,000 MJ, while the GPU cost is 0.8MJ.

(2) The training process is notably more energy-intensive on the CPU compared to the GPU, as there can be a considerable amount of CPU-GPU communication that is also counted on the CPU cost side as CPU controls the communication. This difference can be even more significant for applications that sample on the CPU side. For the NLB, we observed an additional 2,500MJ CPU energy and 12MJ GPU energy for training on the Wikitalk dataset. For TGN-uniform, it requires 10,000MJ CPU energy and 10MJ GPU energy in extra.

I The efficiency evaluation of different sampling methods

In this section, we give additional details to show that forward sampling is indeed more efficient than samplings used by the conventional TGRL methods in practice. During the inference phase, both the proposed forward sampling and truncation methods can retrieve the down-sampled neighbors of a node in $O(1)$ time, while uniform sampling takes $O(N_u^t)$ time, where N_u^t is the number of neighbors of node u at time t. Since we typically need to infer a batch of nodes at a time, we prefer the sampling for the entire batch to be conducted in parallel. Both truncation and uniform sampling require storing and sampling from historical interactions. Due to the uneven distribution of degrees among different nodes, these operations can only be carried out within the CPUs, even when performed in parallel. On the contrary, our method can directly retrieve the already sampled neighbors within the GPU in constant time.

[13] https://pyjoules.readthedocs.io/en/latest/

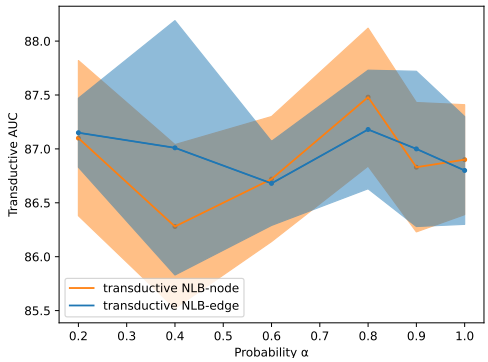

Figure 5: The changes in transductive link prediction test performance with respect to different α on the Ubuntu dataset.

dataset	proposed sampling	trunc. 1-thread	trunc. 32-thread	unif. 1-thread	unif. 32-thread
Wiki-talk	0.19	6.06	5.47	8.90	6.53
Reddit	0.10	0.96	0.68	1.02	0.82
GDELT	0.55	119.19	93.46	390.24	181.47

Table 12: The accumulated sampling time for inference during the training process over one epoch in seconds.

We have conducted an experiment that measures only the accumulated sampling time during inference for one epoch in seconds for three datasets (Table 12). The results show that our method can retrieve the samples instantaneously, while both truncation and uniform sampling are significantly slower in comparison. Uniform sampling is even slower than truncation, and multi-thread (32 threads) sampling is faster than single-thread sampling, especially for large datasets, as expected.

	task	dataset	0.2	0.4	0.6	0.8	0.9	1.0
NLB-edge	trans.	Reddit	72.39 ± 2.15	72.86 ± 2.06	73.13 ± 3.44	73.47 ± 2.48	75.40 ± 1.85	72.77 ± 2.69
NLB-node			66.21 ± 2.95	64.85 ± 3.83	67.27 ± 2.00	68.15 ± 1.45	67.65 ± 5.31	67.85 ± 2.72
NLB-edge	induc.	Ubuntu	21.59 ± 4.83	21.74 ± 2.84	22.72 ± 3.35	23.84 ± 2.27	21.13 ± 2.44	21.49 ± 4.49
NLB-node			22.73 ± 3.07	21.08 ± 3.01	20.60 ± 4.22	20.54 ± 3.48	21.46 ± 2.97	21.30 ± 4.92

Table 13: The performance measured in MRR with 500 negative samples of NLB-edge and NLB-node given different α's. $\alpha = 0.9$ is the setting used for comparison with baselines.

	task	0	5	10	15	20	25	30
NLB-edge	trans.	62.35 ± 1.30	70.62 ± 3.16	69.93 ± 4.06	73.13 ± 1.98	75.40 ± 1.85	73.52 ± 1.34	72.30 ± 3.38
NLB-node		62.35 ± 1.30	66.14 ± 4.74	68.96 ± 1.67	69.17 ± 1.82	67.65 ± 5.31	67.10 ± 4.42	66.78 ± 1.94
NLB-edge	induc.	36.59 ± 3.49	54.65 ± 4.36	59.62 ± 1.03	56.09 ± 5.45	61.74 ± 1.85	61.87 ± 2.03	61.33 ± 2.65
NLB-node		36.59 ± 3.49	47.69 ± 6.44	51.15 ± 5.36	54.50 ± 4.76	53.43 ± 8.03	57.06 ± 7.02	58.09 ± 0.93

Table 14: The performance measured in MRR with 500 negative samples of NLB-edge and NLB-node on the Reddit dataset given different down-sampled neighbor hash table sizes s. $s = 20$ is the setting used for comparison with baselines.