UTG: Towards a Unified View of Snapshot and Event Based Models for Temporal Graphs

Shenyang Huang[1,2,*] **Farimah Poursafaei**[1,2,*]
Reihaneh Rabbany[1,2,5] **Guillaume Rabusseau**[1,4,5] **Emanuele Rossi**[3]
[1]Mila - Quebec AI Institute, [2]School of Computer Science, McGill University
[3]VantAI, [4]DIRO, Université de Montréal, [5]CIFAR AI Chair

Abstract

Many real world graphs are inherently dynamic, constantly evolving with node and edge additions. These graphs can be represented by temporal graphs, either through a stream of edge events or a sequence of graph snapshots. Until now, the development of machine learning methods for both types has occurred largely in isolation, resulting in limited experimental comparison and theoretical cross-pollination between the two. In this paper, we introduce Unified Temporal Graph (UTG), a framework that unifies snapshot-based and event-based machine learning models under a single umbrella, enabling models developed for one representation to be applied effectively to datasets of the other. We also propose a novel UTG training procedure to boost the performance of snapshot-based models in the streaming setting. We comprehensively evaluate both snapshot and event-based models across both types of temporal graphs on the temporal link prediction task. Our main findings are threefold: first, when combined with UTG training, snapshot-based models can perform competitively with event-based models such as TGN and GraphMixer even on event datasets. Second, snapshot-based models are at least an order of magnitude faster than most event-based models during inference. Third, while event-based methods such as NAT and DyGFormer outperforms snapshot-based methods on both types of temporal graphs, this is because they leverage joint neighborhood structural features thus emphasizing the potential to incorporate these features into snapshot-based models as well. These findings highlight the importance of comparing model architectures independent of the data format and suggest the potential of combining the efficiency of snapshot-based models with the performance of event-based models in the future.

1 Introduction

Recently, Graph Neural Networks (GNNs)[1, 2] and Graph Transformers[3, 4] have achieved remarkable success in various tasks for static graphs, such as link prediction, node classification, and graph classification [5]. These successes are driven by standardized empirical comparisons across model architectures [5] and theoretical insights into the expressive power of these models [6].

However, real-world networks such as financial transaction networks [7], social networks [8], and user-item interaction networks [9] are constantly evolving and rarely static. These evolving networks are often modeled by Temporal Graphs (TGs), where entities are represented by nodes and temporal relations are represented by timestamped edges between nodes. Temporal graphs are categorized into two types: Discrete-Time Dynamic Graphs (DTDGs) and Continuous-Time Dynamic Graphs (CTDGs) [10]. DTDGs are represented by an ordered sequence of graph snapshots, while CTDGs

[*]Equal contributions.

consist of timestamped edge streams. Both representations of temporal graphs are prevalent in real-world applications.

Until now, the development of ML methods for both types has occurred mostly independently, resulting in limited experimental comparison and theoretical cross-pollination between the two. We argue that the time granularity of the data collection process together with the requirements of the downstream task have created a gap between DTDG and CTDG in *model development* and *evaluation*.

Isolated Model Development. Despite the similarities between DTDGs and CTDGs, models for these graphs have been developed largely in isolation. Adopting the terminology of [11], models targeting DTDGs focus on learning from a sequence of graph snapshots (*snapshot-based models*) [12–14] , while methods for CTDGs focus on learning from a stream of timestamped edge events (*event-based models*) [15–17]. The disparate data representations of DTDGs and CTDGs have impeded comprehensive comparison across models developed for each category. Consequently, there are limited theoretical insights and empirical evaluations of the true potential of these models when compared together. In real-world applications, representing the data as CTDGs or DTDGs is often a design choice, and the ambiguity of the actual performance merits of both categories makes it challenging to select the optimal model in a practical setting.

Distinct Evaluation Settings. Another obstacle to comparing snapshot and event-based methods is their distinct evaluation settings. Snapshot-based methods have been primarily tested under the *deployed setting* [12, 14] [2], where the test set information is strictly not available to the model, and training set information is used for prediction. In contrast, event-based models are designed for the *streaming setting* [15, 19], where streaming predictions allow the model to use recently observed information, enabling event-based models to update their node representations at test time.

In this work, we aim to bridge the gap between event-based and snapshot-based models by providing a unified framework to train and evaluate them to predict future events on any type of temporal graph. Our main contributions are as follows:

- **Unified framework**: We propose *Unified Temporal Graph* (UTG), a framework that unifies snapshot-based and event-based temporal graph models under a single umbrella, enabling models developed for one representation to be applied effectively to datasets of the other.
- **Updating snapshot-based models**: We propose a novel UTG training strategy to boost the performance of snapshot-based models in the streaming setting. This allows snapshot-based models to achieve competitive performance with event-based models such as TGN and GraphMixer on the `tgbl-wiki` and Reddit CTDG datasets.
- **Benchmarking**: By leveraging the UTG framework, we conduct the first systematic comparison between snapshot and event-based models on both CTDG and DTDG datasets. While some event-based methods such as NAT and DyGFormer outperform snapshot-based methods on both CTDGs and DTDGs, we posit this is due to leveraging joint neighborhood structural features rather than a fundamental property of event-based methods. Additionally, snapshot-based methods are at least an order of magnitude faster than event-based methods while achieving competitive performance. This suggests several future directions, such as integrating joint neighborhood structural features in snapshot-based models and developing a universal method that combines accuracy and efficiency for both DTDGs and CTDGs.

Reproducibility: The code and data for this project is publicly available on Github:
`https://github.com/shenyangHuang/UTG`

2 Related Work

Holme et al. [20] provided a general overview of the many types of real world temporal networks showing that temporal graphs are ubiquitous in many applications. Recently, many ML methods were developed for temporal graphs. The well-adopted categorization by Kazemi et al. [10] are defined by the two types of temporal graphs: Discrete Time Dynamic Graphs (DTDGs) and Continuous Time Dynamic Graphs (CTDGs). Due to difference in input data format, empirical comparison between methods designed for CTDGs and DTDGs are under-explored and these two categories are often

[2]An exception to this is ROLAND [18], which proposed the live-update setting (see Appendix B for a more detailed discussion).

considered distinct lines of research despite many similarities in models' design. More detailed discussions on related work can be found in Appendix C.

Discrete Time Dynamic Graphs. Early methods often represent temporal graphs as a sequence of graph snapshots while adapting common graph neural networks such as Graph Covolution Network (GCN) [1] used in static graphs for DTDGs. For example, EGCN [14] employs a Recurrent Neural Network (RNN) to evolve the parameters of a GCN over time. In comparison, GCLSTM [13] learns the graph structure via a GCN while capturing temporal dependencies with an LSTM network [21]. Pytorch Geometric-Temporal (PyG Temporal) [22] is a comprehensive framework that facilitate neural spatiotemporal signal processing which implements existing work such as EGCN and GCLSTM in an efficient manner. HTGN [12] utilizes hyperbolic geometry to better capture the complex and hierarchical nature of the evolving networks. Recently, You et al. [18] introduced a novel *live-update setting* where GNNs are always trained on the most recent observed snapshot after making predictions. In comparison, the *streaming setting* in this work allows the model to use observed snapshots for forward pass but no training are permitted on test set. Zhu et al. [23] designed the WinGNN framework for the live-update setting where a simple GNN with meta-learning strategy is used in combination with a novel random gradient aggregation scheme, removing the need for temporal encoders.

Continuous Time Dynamic Graphs. Event-based methods process temporal graphs as a stream of timestamped edges. DyRep [24] and JODIE [9] are two pioneering work on CTDGs. TGAT [25] is one of the first work for studying inductive representation learning on temporal graphs. Rossi et al. [15] introduce Temporal Graph Networks (TGNs), a generic inductive framework of Temporal Graph Networks, showing DyRep, JODIE and TGAT as its special cases. Methods such as CAWN [26] and NAT [17] both focuses on learning the joint neighborhood of the two nodes of interest in the link prediction task. CAWN focuses on learning from temporal random walks while NAT is a neighborhood-aware temporal network model that introduces a dictionary-type neighborhood representation for each node. TCL [27] and DyGFormer [28] applies transformer based architecture on CTDGs, inspired by the success of transformer based architectures on time series [29], images [30] and natural language processing [31]. Despite the promising performance of event-based methods, recent work showed significant limitations in the standard link prediction evaluation due to the simplicity of negative samples used for evaluation [16]. To improve the evaluation for CTDG, Huang et al. [19] proposed the Temporal Graph Benchmark (TGB), a collection of large-scale and realistic datasets from distinct domains for both link and node level tasks.

3 Preliminaries

Definition 1 (Continuous Time Dynamic Graphs) *A Continuous Time Dynamic Graph (CTDG) $\mathcal{G}$ is formulated as a collection of edges represented as tuples with source node, destination node, start time and end time;*

$$\mathcal{G} = \{(s_0, d_0, t_0^s, t_0^e), (s_1, d_1, t_1^s, t_1^e), \dots, (s_k, d_k, t_k^s, t_k^e)\}$$

where, for edge $i \in [0, k]$, s_i and d_i denote source and destination respectively. The start times are ordered chronologically $t_0^s \leq t_1^s \leq \dots \leq t_k^s$, each start time is less than or equal to the corresponding end time $t_i^s \leq t_i^e$, hence for each timestamp t we have $t \in [t_0^s, t_k^e]$.

Without loss of generality, one can normalize the timestamps in $\mathcal{G}$ from $[t_0, t_k]$ to $[0, 1]$ by applying $t = \frac{t-t_0}{t_k} \forall t \in [t_0, t_k]$. Real world temporal networks can be broadly classified into two inherent types based on the nature of their edges: *transient networks* and *persistent networks*. Examples of spontaneous networks include transaction networks, retweet networks, Reddit networks, and other activity graphs. Here, the edges are spontaneous thus resulting in the start time and end time of an edge being the same, i.e. $t^s = t^e$. This formulation is inline with related studies in [10, 16, 19, 32]. For relationship networks such as friendship networks, contact networks, and collaboration networks, the edges often persist over a period of time resulting in $t^s \neq t^e$.

Definition 2 (Discrete Time Dynamic Graphs) *A Discrete Time Dynamic Graph (DTDG) $\mathbf{G}$ is a sequence of graph snapshots sampled at regularly-spaced time intervals [10]:*

$$\mathbf{G} = \{\mathbf{G}_0, \mathbf{G}_1, \dots, \mathbf{G}_T\}$$

$\mathbf{G}_t = \{\mathbf{V}_t, \mathbf{E}_t\}$ *is the graph at snapshot $t \in [0, T]$, where $\mathbf{V}_t$, $\mathbf{E}_t$ are the set of nodes and edges in $\mathbf{G}_t$, respectively.*

4 UTG Framework

In this section, we present the Unified Temporal Graph (UTG) framework which aims to unify snapshot-based and event-based temporal graph models under the same framework, enabling temporal graph models to be applied to both CTDGs and DTDGs. UTG has two key components: *input mapper* and *output mapper*. Input mapping converts the input temporal graph into the appropriate representation needed for a given method, i.e. snapshots or events. Output mapper transforms the prediction of the model to the required time granularity of the task. Figure 1 shows the workflow of UTG framework. UTG enables any temporal graph learning methods to be applied to any input temporal graph.

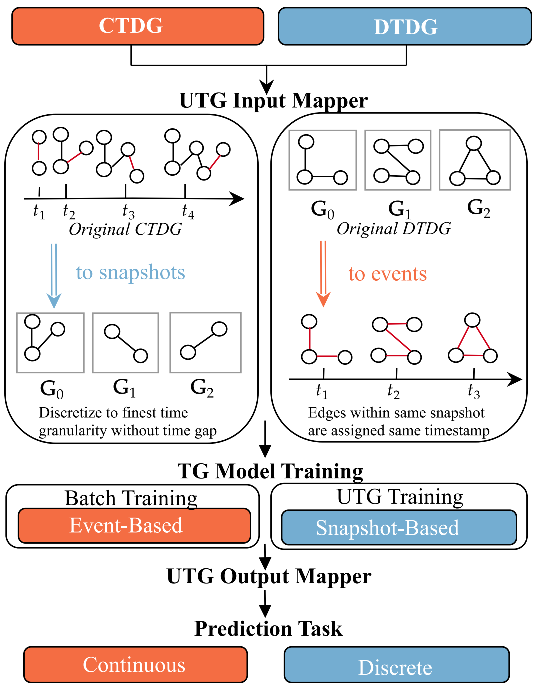


Figure 1: Illustration of the UTG framework. The input graph is processed by the UTG input mapper to generate the appropriate input data format for TG models. The model predictions are then processed by the UTG output mapper for prediction.

4.1 UTG Input Mapper

Both snapshot and event-based TG methods require specific input data format. For snapshot-based models, discretizing CTDG data into a sequence of snapshots is required. For event-based models, DTDG snapshots need to be converted into batches of events.

Converting CTDG to Snapshots. Here, we formulate the discretization process which converts a continuous-time dynamic graph into a sequence of graph snapshots for snapshot-based models.

Definition 3 (Discretization Partition) *Let* 0 *and* 1 *be the normalized start and end time of a temporal graph $\mathcal{G}$. A discretization partition* $\mathbf{P}$ *of the interval* $[0, 1]$ *is a collection of intervals:*

$$\mathbf{P} = \{[\tau_0, \tau_1], [\tau_1, \tau_2], \dots, [\tau_{k-1}, \tau_k]\}$$

such that $0 = \tau_0 < \tau_1 < \cdots < \tau_k = 1$ *and where* $k \in \mathbb{N}$.

Hence, a discretization partition $\mathbf{P}$ defines a finite collection of non-overlapping intervals and its norm is defined as:

$$||\mathbf{P}|| = \max\{|\tau_1 - \tau_0|, |\tau_2 - \tau_1|, \dots, |\tau_k - \tau_{k-1}|\}$$

The norm $||\mathbf{P}||$ can also be interpreted as the *max duration* of a snapshot in the temporal graph $\mathcal{G}$. The cardinality of $\mathbf{P}$ is denoted by $|\mathbf{P}|$.

Definition 4 (Regular Discretization Partition) *A given discretization partition* $\mathbf{P}$ *is regular if and only if:*

$$\forall [\tau_i, \tau_j] \in \mathbf{P},\ |\tau_j - \tau_i| = ||\mathbf{P}|| = \frac{|\tau_k - \tau_0|}{|\mathbf{P}|}$$

In this case, all intervals have the same duration equal to $||\mathbf{P}||$.

Definition 5 (Induced Graph Snapshots) *Given a Continuous Time Dynamic Graph $\mathcal{G}$ and a Regular Discretization Partition* $\mathbf{P}$, *the Induced Graph Snapshots* $\mathbf{G}$ *are formulated as:*

$$\mathbf{G} = \{\mathbf{G}_{\tau_0}^{\tau_1}, \mathbf{G}_{\tau_1}^{\tau_2}, \dots, \mathbf{G}_{\tau_{k-1}}^{\tau_k}\}$$

where $\mathbf{G}_{\tau_i}^{\tau_j}$ *is defined as the aggregated graph snapshot containing all edges that have a* start time $t_s < \tau_j$ *and an* end time $t_e \geq \tau_i$, *i.e. edges that are present solely within the* $[\tau_i, \tau_j)$ *interval.*

Note that for spontaneous networks, each edge exist at a specific time point $t_s = t_e$ thus only belonging to a single interval/snapshot. For relationship networks however, it is possible for an edge to belong to multiple intervals depending on its duration.

Definition 6 (Discretization Level) *Given a regular discretization partition* $\mathbf{P}$ *and the timestamps in a temporal graph* $\mathcal{G}$ *normalized to* $[0, 1]$*, the* discretization level Δ *of* $\mathbf{P}$ *is computed as :*

$$\Delta = \frac{1}{|\mathbf{P}|}$$

where $|\mathbf{P}|$ *is the cardinality of the partition or the number of intervals.*

Note that $\Delta \in [0, 1]$. When $|\mathbf{P}| = 1$ then $\Delta = 1$ which means the temporal graph is collapsed into a single graph snapshot (i.e. a static graph). On the other extreme, we have $\lim_{|\mathbf{P}|\to\infty} \Delta = 0$, preserving the continuous nature of the continuous time dynamic graph $\mathcal{G}$.

Definition 7 (Time Gap) *Given a Continuous Time Dynamic Graph* $\mathcal{G}$ *and a Regular Discretization Partition* $\mathbf{P}$*, a Time Gap occurs when there exist one or more snapshots in the Induced Graph Snapshots* $\mathbf{G}$ *with an empty edge set.*

In this work, we choose the number of intervals in discretization by selecting the finest time granularity which would not induce a time gap. This ensures that there are no empty snapshots in the induced graph snapshots.

Converting DTDG to Events. While it may seem straightforward to convert DTDG to events — simply create one event with timestep t for each edge in snapshot G_t — some subtleties related to batch training and memory update of event-based models have to be considered to avoid data leakage. Event-based models often receive batches of events (or edges) with a fixed dimension as inputs [15, 17, 28]. In discrete-time dynamic graphs, all edges in a snapshot have the same timestamp and are assumed to arrive simultaneously. Therefore, using a fixed batch size can result in splitting the snapshot into multiple batches. Because models in the *streaming setting* [19] such as TGN [15] and NAT [17] update their representation of the temporal graph at the end of each batch, predicting a snapshot across multiple batches leads to data leakage: a portion of the edges from the snapshot is used to predict other (simultaneous) edges from the same snapshot. To avoid data leakage on DTDGs, we ensure that each snapshot is contained in a single batch for event-based models[3]. Note that the issue of splitting edges that share the same timestamp into multiple batches can exist in general CTDG datasets as well, more likely for datasets with a large burst of edges at a single timestamp.

4.2 UTG Output Mapper

The output task on the temporal graph can be either discrete or continuous. Discrete tasks refer to predicting which edges will be present at a future snapshot (with an integer timestep). Continuous tasks refer to predicting which edges will be present for a given UNIX timestamp in the future. Snapshot-based models often omits the timestamp of the prediction as an input, implicitly assuming the prediction is for the next snapshot. Therefore, applying snapshot-based models for a continuous task requires 1). always updating the model with all the information available until the most recent observed snapshot (*test-time update*) and 2). mapping the discrete-time prediction to a continuous timestamp. We explain here how to map the prediction to a continuous space with zero-order hold.

Definition 8 (Zero-order Hold) *A discrete time signal* $y[i]$*,* $i \in \mathbb{N}$*, can be converted to a continuous time signal* $y(t)$*,* $t \in \mathbb{R}$*, by broadcasting the value y[i] as a constant in the interval* $[\tau_i, \tau_j]$*:*

$$y(t) = y[i], \ \textit{for all} \quad \tau_i \le t \le \tau_j$$

where $[\tau_i, \tau_j]$ *specifies the duration of the discrete signal.*

By applying zero-order hold for snapshot-based models, the predictions can now be broadcasted for a period of time (specifically for the duration of a given snapshot $[\tau_i, \tau_j]$). Therefore, it is now possible to utilize snapshot-based models on continuous-time dynamic graphs. Note that often

[3]In case the resulting batch would not fit in memory, one can delay the memory update (and parameter updates during training) only after all edges from the current snapshot have been processed, something akin to gradient accumulation.

snapshot-based model are designed to predict for the immediate next snapshot and not capable of predicting snapshots in more distant future. Therefore, inherently their ability to predict events in the far future is limited when compared to event-based model that explicitly takes a timestamp as input. With zero-order hold, we assume that the event to predict next is within the next snapshot to circumvallate the aforementioned limitation of snapshot-based model. To achieve this, we select the finest time granularity on the CTDG datasets which results in no time gap to construct snapshots.

4.3 Streaming Evaluation

Event-based models often evaluate with the *streaming setting* [15, 17, 26, 28]. In this setting, information from the previously observed batches of events (or graph snapshot) can be used to update the model however no information from the test set is used to train the model. In comparison, snapshot-based models are often evaluated in either the *live-update setting* [18] or the *deployed setting* [12, 14]. Detailed differences between evaluation settings are discussed in Appendix B.

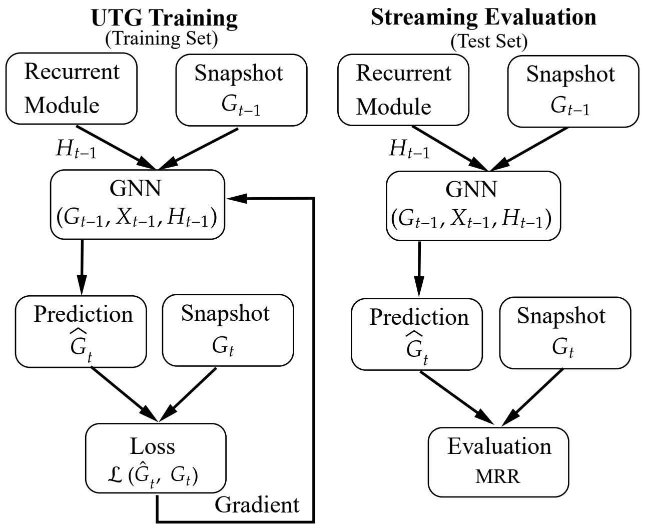


Figure 2: UTG training and evaluation workflow. UTG training enables snapshot-based models to perform better for the streaming setting.

In this work, to provide a unified comparison, we focus on the widely used streaming setting for experimental evaluation as it closely resembles real-world settings where after the model is trained, it is required to incorporate newly observed information into its predictions. Note that the UTG framework can also be applied to test under the deployed setting with little changes. For the live-update setting, the changes are required for each model's training procedure as the training set is not split chronologically but rather being a subgraph of each observed snapshot. Figure 2 shows the evaluation pipeline used for snapshot-based models in UTG. After a snapshot is observed, it can be used to update the node representation of the snapshot-based models for the prediction of the next snapshot (only forward pass for inference).

4.4 UTG Training for Snapshot-based Models

Here, we discuss the changes to the snapshot-based methods in the UTG framework. Figure 2 illustrates the work flow of UTG training for snapshot-based models. The standard training for snapshot-based models, e.g. with Pytorch Geometric Temporal [33], are designed for tasks such as graph regression or node classification, for the deployed setting. Therefore, the training procedure needs to be adapted accordingly for the link prediction with the streaming setting.

Difference of UTG Training. The first required change is that in standard training, the snapshots up until time t is used as input for predictions at time t. This is feasible for node classification and graph regression tasks: the graph structure at time t is used to predict the unknown target labels. However, this is problematic for the link prediction task as the target itself (the graph structure) is not available as input to the model. Therefore, to account for this, we modified the training to use only snapshots up to $\mathbf{G}_{t-1}$ as input to predict the graph structure at the current step $\mathbf{G}_t$.

The second change is that, in UTG, the loss is backpropagated to the model at each snapshot. This is in contrast with accumulating the loss through the whole sequence and only backpropagation once at the end of training (as seen in standard training). This change is motivated by the training procedure often seen in event-based models for the streaming setting. For example, in TGN [15], loss on each batch is computed based on information from the previous batch. UTG training enhances the performance of snapshot-based models for the streaming setting and in Section 5, we demonstrate the performance advantage of UTG training on a number of snapshot-based models.

Connection to Truncated Backprop Through Time. By considering the temporal graph as a sequence (of snapshots), the link prediction problem can be seen as a time series prediction task

Table 1: Dataset statistics.

	Dataset	# Nodes	# Edges	# Unique Edges	Surprise	Time Granularity	# Snapshots
DTDG	UCI	1,899	26,628	20,296	0.535	Weekly	29
	Enron	184	10,472	3,125	0.253	Monthly	45
	Contact	694	463,558	79,531	0.098	Hourly	673
	Social Evo.	74	87,479	4,486	0.005	Daily	244
	MOOC	7,144	236,808	178,443	0.718	Daily	31
CTDG	`tgbl-wiki`	9,227	157,474	18,257	0.108	UNIX timestamp	745 (Hourly)
	`tgbl-review`	352,637	4,873,540	4,730,223	0.987	UNIX timestamp	237 (Monthly)
	Reddit	10,984	672,447	78,516	0.069	UNIX timestamp	745 (Hourly)

where the information until time $(t-1)$ is used to predict for time t. Many snapshot-based models utilize a RNN to model temporal dependency. In this view, accumulating gradients throughout the whole sequence before backpropagation is equivalent to the classical backpropagation through time algorithm [34]. From this perspective, backpropagating the loss at each snapshot in UTG training can be interpreted as using the Truncated Backpropagation Through Time (TBTT) algorithm to train the model, with a window size of one. It is known that TBTT helps circumvent common issues of training RNNs such as exploding memory usage and vanishing gradient problem [35, 36]. Therefore, UTG training might help alleviate the vanishing gradient problem.

5 Experiments

In this section, we benchmark both snapshot-based and event-based methods across both CTDG and DTDG datasets under the UTG framework.

Datasets. In this work, we consider five discrete-time dynamic graph datasets and three continuous-time dynamic graph datasets. `tgbl-wiki` and `tgbl-review` are datasets from TGB [19] while the rest are found in Poursafaei et al. [16]. The dataset statistics are shown in Table 1. The time granularity or discretization level of each DTDG dataset is selected as the finest time granularities where there are no time gaps. The surprise index is defined as $surprise = \frac{|E_{\text{test}} \setminus E_{\text{train}}|}{E_{\text{test}}}$ [16] which measures the proportion of unseen edges in the test set when compared to the training set.

Evaluation Setting. A common approach for evaluating dynamic link prediction tasks is similar to binary classification, where one negative edge is randomly sampled for each positive edge in the test set, and performance is measured using metrics like the Area Under the Receiver Operating Characteristic curve (AUROC) or Average Precision (AP) [15, 25]. However, recent studies have shown that such evaluation is overly simplistic and tends to inflate performance metrics for most models [16, 19]. One main reason is that randomly sampled negative edges are too easy and the more challenging *historical negatives* (past edges absent in the current timestamp) are rarely sampled [16]. To address these issues, several improvements have been proposed, framing the problem as a ranking task where the model must identify the most probable edge from a large pool of negative samples as well as adding challenging negatives. Therefore, we adopt the improved evaluation methodology used in TGB [19], where link prediction is treated as a ranking problem and the Mean Reciprocal Rank (MRR) metric is applied. This metric calculates the reciprocal rank of the true destination node among a large number of possible destinations.

For each dataset, we generate a fixed set of negative samples for each positive edge consisting of 50% *historical negatives* and 50% *random negative*, same as in [19]. For DTDG datasets, we generate 1000 negative samples per positive edge. For TGB datasets, we use the same set of negatives provided in TGB and for Reddit, we generate 1000 negatives similar to before. For graphs with less than 1k nodes, we generate negative samples equal to number of nodes. We follow the *streaming setting* where the models are allowed to update their representation at test time while gradient updates are not permitted. For DTDG datasets, we select the best results from learning rate 0.001 or 0.0002. For CTDG datasets, we report the results from TGB [19] where available or by learning rate 0.0002.

Compared Methods. We compare four event-based methods including TGN [15], DyGFormer [28], NAT [17] and GraphMixer [37]. We also include Edgebank [16], a scalable and non-parameteric heuristics. In addition, we compare three existing snapshot-based methods including HTGN [12], GCLSTM [13], EGCNo [14], and ROLAND-GRU [18]. Lastly, we adapt a common 2-layer (static) GCN [1] under the UTG framework to demonstrate the flexibility of UTG (without a recurrent

Table 2: Test MRR comparison for snapshot and event-based methods on DTDG datasets, results reported from 5 runs. Top three models are marked by **First**, Second, *Third*.

	Method	UCI	Enron	Contacts	Social Evo.	MOOC
event	TGN [15]	0.091 ± 0.002	0.191 ± 0.027	0.153 ± 0.007	0.283 ± 0.009	0.174 ± 0.009
	DyGFormer [28]	0.334 ± 0.024	**0.331** ± 0.010	**0.283** ± 0.006	**0.366** ± 0.004	OOM
	NAT [17]	**0.356** ± 0.048	*0.276* ± 0.014	0.245 ± 0.015	0.258 ± 0.036	**0.283** ± 0.058
	GraphMixer [37]	0.105 ± 0.008	0.296 ± 0.019	0.055 ± 0.003	0.157 ± 0.005	OOM
	EdgeBank$_\infty$ [16]	0.055	0.115	0.016	0.049	0.040
	EdgeBank$_{\text{tw}}$ [16]	*0.165*	0.157	0.050	0.070	0.070
snapshot	HTGN (UTG) [12]	0.093 ± 0.012	0.267 ± 0.007	0.165 ± 0.001	0.228 ± 0.003	0.093 ± 0.005
	GCLSTM (UTG) [13]	0.093 ± 0.006	0.170 ± 0.008	0.128 ± 0.004	*0.286* ± 0.003	*0.143* ± 0.006
	EGCNo (UTG) [14]	0.121 ± 0.010	0.233 ± 0.008	*0.192* ± 0.001	0.253 ± 0.006	0.126 ± 0.009
	GCN (UTG) [1]	0.068 ± 0.009	0.164 ± 0.011	0.104 ± 0.002	0.289 ± 0.008	0.084 ± 0.010
	ROLAND (UTG) [18]	0.103 ± 0.011	0.243 ± 0.017	0.145 ± 0.002	0.240 ± 0.005	0.121 ± 0.003

module). If a method runs out of memory on a NVIDIA A100 GPU (40GB memory), it is reported as out of memory (OOM). If a method runs for more than 5 days, it is reported as out of time (OOT).

With the UTG framework, we can now compare snapshot-based methods and event-based methods on any temporal graph dataset. This comparison allows us to focus on analyzing the strength and weaknesses of the model design, independent of the data format.

DTDG Results. Table 2 shows the performance of all methods on the DTDG datasets. Surprisingly, we find that event-based methods achieve state-of-the-art performance on the DTDG datasets, particularly with DyGFormer and NAT consistently outperforming other methods. With the improvements from UTG, snapshot-based models can obtain competitive performance on datasets such as Enron and Social Evo. Interestingly, even the simple GCN with UTG training can achieve second place performance on the Social Evo. dataset. Note that this dataset has the lowest surprise out of all datasets meaning the majority of test set edges have been observed during training, possibly explaining the strong performance of GCN in this case. Lastly, on the MOOC dataset which has the largest number of nodes out of all DTDG datasets, both DyGformer and GraphMixer ran out of memory (OOM) showing their difficulty in scaling with the number of nodes in a snapshot.

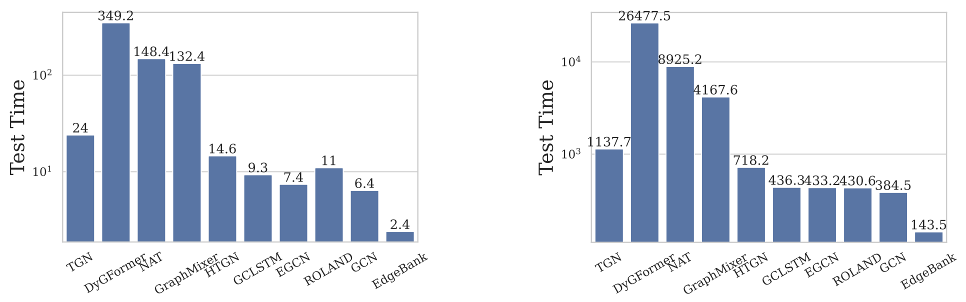
CTDG Results. Table 3 shows the performance of all methods on the CTDG datasets. Similar to DTDG datasets, DyGformer and NAT retains competitive performance here. On the `tgbl-wiki` and Reddit dataset, HTGN, a snapshot-based model is able to outperform widely-used TGN and GraphMixer model. This shows that learning from the discretized snapshots can be effective even on CTDG datasets. However, snapshot-based models have lower performance on the `tgbl-review` dataset where the surprise index is high. This shows that the inductive reasoning capability on snapshot-based models should be further improved to generalize to unseen edges.

Table 3: Test MRR comparison for snapshot and event-based methods on CTDG datasets, results reported from 5 runs. Top three models are marked by **First**, Second, *Third*.

	Method	`tgbl-wiki`	`tgbl-review`	Reddit
event	TGN [15]	0.396 ± 0.060	0.349 ± 0.020	0.499 ± 0.011
	DyGFormer [28]	**0.798** ± 0.004	0.224 ± 0.015	OOT
	NAT [17]	0.749 ± 0.010	*0.341* ± 0.020	**0.693** ± 0.015
	GraphMixer [37]	0.118 ± 0.002	**0.521** ± 0.015	0.136 ± 0.078
	EdgeBank$_\infty$ [16]	0.495	0.025	0.485
	EdgeBank$_{\text{tw}}$ [16]	*0.571*	0.023	0.589
snapshot	HTGN (UTG) [12]	0.464 ± 0.005	0.104 ± 0.002	*0.533* ± 0.007
	GCLSTM (UTG) [13]	0.374 ± 0.010	0.095 ± 0.002	0.467 ± 0.004
	EGCNo (UTG) [14]	0.398 ± 0.007	0.195 ± 0.001	0.321 ± 0.009
	GCN (UTG) [1]	0.336 ± 0.009	0.186 ± 0.002	0.242 ± 0.005
	ROLAND (UTG) [18]	0.289 ± 0.003	0.297 ± 0.006	0.211 ± 0.006

Computational Time Comparison Figure 3a and Figure 3b shows the test inference time comparison for all methods on the Social Evo. and the `tgbl-review` dataset respectively. The test time for each dataset is reported in Appendix A. Overall, we observe that snapshot-based methods are at least an order of magnitude faster than most event-based methods. In comparison, high performing model such as DyGformer has significantly higher computational time thus limiting its scalability to large datasets. One promising direction is to combine the performance of event-based models such as NAT and DyGFormer with that of the efficiency of the snapshot-based models for improved scalability.

Benefits of UTG Training. UTG training enables snapshot-based models to effectively incorporate novel information during inference. Table 4 shows the performance benefit of using UTG training on the GCLSTM, HTGN and EGCNo models when compared to the original training scheme (such as



(a) Test time (seconds) for Social Evo. dataset. **(b)** Test time (seconds) for `tgbl-review` dataset.

Figure 3: Snapshot-based models with UTG training are at least an order of magnitude faster than event-based models for inference.

Table 4: Base models with UTG training when compared to original training, results averaged across five runs, best results for each model are **bolded**.

Dataset	GCLSTM (UTG)	GCLSTM (original)	EGCNo (UTG)	EGCNo (original)	HTGN (UTG)	HTGN (original)
UCI	**0.093** ± 0.006	0.047 ± 0.006	**0.121** ± 0.010	**0.124** ± 0.007	**0.093** ± 0.012	0.052 ± 0.006
Enron	**0.170** ± 0.006	0.131 ± 0.003	**0.233** ± 0.008	0.227 ± 0.012	**0.267** ± 0.007	0.196 ± 0.025
Contacts	**0.128** ± 0.004	0.101 ± 0.031	0.192 ± 0.001	**0.195** ± 0.001	**0.165** ± 0.001	0.129 ± 0.020
Social Evo.	**0.286** ± 0.003	**0.287** ± 0.009	**0.253** ± 0.006	0.231 ± 0.009	**0.228** ± 0.003	0.178 ± 0.024
MOOC	**0.143** ± 0.006	0.076 ± 0.003	**0.126** ± 0.009	0.119 ± 0.009	**0.093** ± 0.005	0.079 ± 0.011

in Pytorch Geometric Temporal) for DTDG datasets. UTG training significantly enhances model performance across most datasets and performs identically on the rest.

Discussion. Event-based methods such as NAT and DyGFormer tend to perform best on both CTDG and DTDG, potentially leading to the premature conclusion that event-based modeling is the preferred paradigm and that snapshot-based models should be avoided. However, the superior performance of NAT and DyGFormer could be primarily due to their ability to leverage joint neighborhood structural features, specifically the common neighbors between the source and destination nodes of a link [17, 28]. This approach has been shown to be fundamental for achieving competitive link prediction on static graphs [38, 39]. In contrast, none of the existing snapshot-based methods incorporate joint neighborhood structural features. Therefore, the performance difference could be mainly attributed to this factor rather than an intrinsic difference between event-based and snapshot-based models. This is suggested by the fact that event-based models such as TGN and GraphMixer, which omits these features, have no clear performance advantage over snapshot-based methods. Moreover, snapshot-based methods are more computationally efficient and might be preferred when efficiency is important. These considerations suggest that both event-based and snapshot-based methods have their own merit. We believe that combining the strengths of both approaches is an important future direction.

6 Conclusion

In this work, we introduce the UTG framework, unifying both snapshot-based and event-based temporal graph models under a single umbrella. With the UTG input mapper and UTG output mapper, temporal graph models developed for one representation can be applied effectively to datasets of the other. To compare both types of methods in the streaming setting for evaluation, we propose the UTG training to boost the performance of snapshot-based models. Extensive experiments on five DTDG datasets and three CTDG datasets are conducted to comprehensively compare snapshot and event-based methods. We find that top performing models on both types of datasets leverage joint neighborhood structural features such as the number of common neighbors between the source and destination node of a link. In addition, snapshot-based models can achieve competitive performance to event-based model such as TGN and GraphMixer while being an order of magnitude faster in inference time. Thus, an important future direction is to combine the strength of both types of methods to achieve high performing and scalable temporal graph learning methods.

Acknowledgements

This research was supported by the Canadian Institute for Advanced Research (CIFAR AI chair program), Natural Sciences and Engineering Research Council of Canada (NSERC) Postgraduate Scholarship Doctoral (PGS D) Award and Fonds de recherche du Québec - Nature et Technologies (FRQNT) Doctoral Award.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016. 1, 3, 7, 8, 13, 14

[2] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 1

[3] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021. 1

[4] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022. 1

[5] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020. 1

[6] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018. 1

[7] Kiarash Shamsi, Friedhelm Victor, Murat Kantarcioglu, Yulia Gel, and Cuneyt G Akcora. Chartalist: Labeled graph datasets for utxo and account-based blockchains. *Advances in Neural Information Processing Systems*, 35:34926–34939, 2022. 1

[8] Amirhossein Nadiri and Frank W Takes. A large-scale temporal analysis of user lifespan durability on the reddit social media platform. In *Companion Proceedings of the Web Conference 2022*, pages 677–685, 2022. 1

[9] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019. 1, 3

[10] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020. 1, 2, 3

[11] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Andrea Passerini, et al. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *Transactions on Machine Learning Research*, 2023. 2

[12] Menglin Yang, Min Zhou, Marcus Kalander, Zengfeng Huang, and Irwin King. Discrete-time temporal network embedding via implicit hierarchical learning in hyperbolic space. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1975–1985, 2021. 2, 3, 6, 7, 8, 13, 14

[13] Jinyin Chen, Xueke Wang, and Xuanheng Xu. Gc-lstm: Graph convolution embedded lstm for dynamic network link prediction. *Applied Intelligence*, pages 1–16, 2022. 3, 7, 8, 13, 14

[14] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370, 2020. 2, 3, 6, 7, 8, 13, 14

[15] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020. 2, 3, 5, 6, 7, 8, 13, 14, 15, 16

[16] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction. *Advances in Neural Information Processing Systems*, 35:32928–32941, 2022. 3, 7, 8, 13, 14, 16

[17] Yuhong Luo and Pan Li. Neighborhood-aware scalable temporal network representation learning. In *Learning on Graphs Conference*, pages 1–1. PMLR, 2022. 2, 3, 5, 6, 7, 8, 9, 13, 14, 15, 16

[18] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 2358–2366, 2022. 2, 3, 6, 7, 8, 13

[19] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems*, 2023. 2, 3, 5, 7

[20] Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88:1–30, 2015. 2

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997. 3

[22] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzman Lopez, Nicolas Collignon, et al. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 4564–4573, 2021. 3, 15

[23] Yifan Zhu, Fangpeng Cong, Dan Zhang, Wenwen Gong, Qika Lin, Wenzheng Feng, Yuxiao Dong, and Jie Tang. Wingnn: Dynamic graph neural networks with random gradient aggregation window. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3650–3662, 2023. 3

[24] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*, 2019. 3

[25] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020. 3, 7, 16

[26] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *International Conference on Learning Representations*. 3, 6, 13, 15

[27] Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. Tcl: Transformer-based dynamic graph modelling via contrastive learning. *arXiv preprint arXiv:2105.07944*, 2021. 3

[28] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. *Advances in Neural Information Processing Systems*, 36: 67686–67700, 2023. 3, 5, 6, 7, 8, 9, 13, 14, 16

[29] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: a survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 6778–6786, 2023. 3

[30] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. 3

[31] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 3

[32] Ingo Scholtes. When is a network a network? multi-order graphical model selection in pathways and temporal networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1037–1046, 2017. 3

[33] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzman Lopez, Nicolas Collignon, and Rik Sarkar. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, page 4564–4573, 2021. 6

[34] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 7

[35] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pages 1183–1188. IEEE, 1993. 7

[36] R Pascanu. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2013. 7

[37] 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 *The Eleventh International Conference on Learning Representations*, 2022. 7, 8, 13, 14

[38] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018. 9

[39] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning, 2020. 9

[40] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. 15

[41] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*, 2020. 15

[42] Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. Tcl: Transformer-based dynamic graph modelling via contrastive learning. *arXiv preprint arXiv:2105.07944*, 2021. 15

A Computational Time

Table 5: Test inference time comparison for snapshot and event based methods on DTDG datasets, we report the average result from 5 runs. Top three models are coloured by **First**, Second, *Third*.

	Method	UCI	Enron	Contacts	Social Evo.	MOOC
event	TGN [15]	1.07	1.71	137.57	24.04	50.04
	DyGFormer [28]	155.58	57.72	15423.99	349.22	OOM
	NAT [17]	3.82	8.39	596.22	148.43	299.00
	GraphMixer [37]	32.88	13.85	3542.88	132.39	OOM
	EdgeBank$_\infty$ [16]	0.52	**0.24**	*45.33*	**2.07**	**5.17**
	EdgeBank$_{\mathrm{tw}}$ [16]	0.52	0.25	50.77	2.45	6.12
snapshot	HTGN (UTG) [12]	0.61	0.87	76.64	14.59	28.64
	GCLSTM (UTG) [13]	**0.35**	0.46	40.83	9.27	19.78
	EGCNo (UTG) [14]	0.43	0.45	**40.62**	7.35	15.49
	GCN (UTG) [1]	*0.50*	*0.31*	56.88	*6.40*	*13.30*

Table 5 shows the inference time for all methods on DTDG datasets. Table 6 shows the inference time for all methods on CTDG datasets. OOM means out of memory and OOT means out of time. We observe that snapshot-based models are at least one order of magnitude faster than event-based models such as NAT, DyGFormer and GraphMixer. In addition, the best performing model on most datasets, DyGFormer, is also consistently the slowest method.

B Evaluation Settings

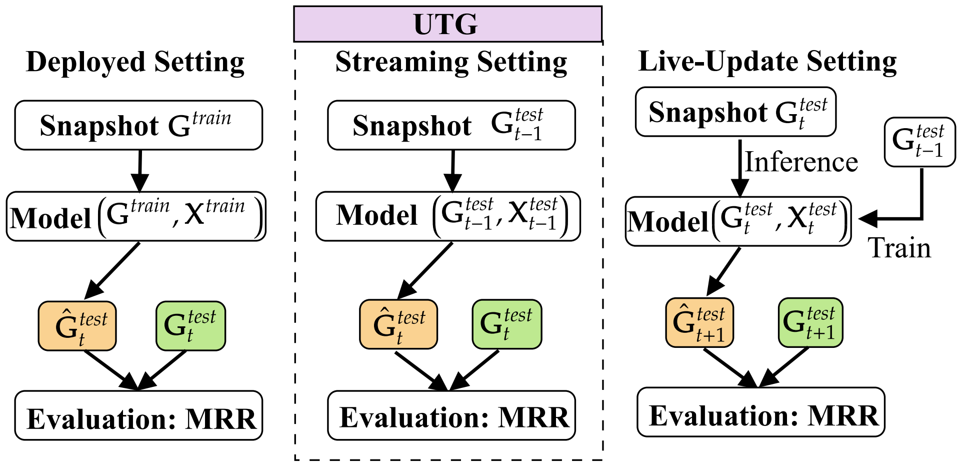


Figure 4: Different setting for evaluation of future link prediction include between *deployed*, *streaming* and *live-update* setting. UTG framework is designed for the streaming setting.

Deployed setting The *deployed setting* is often used as the evaluation setting for snapshot-based methods [12, 14]. In this setting, no information from the test set is passed to the model, and the node embeddings from the last training snapshot are used for predictions in all test snapshots.

Streaming setting event-based models often evaluate with the *streaming setting* [15, 17, 26, 28]. In this setting, information from the previously observed batches of events can be used to update the model however no information from the test set can be used to train the model.

Live-update Setting You et al. [18] proposed the *live-update setting* where the model weights are constantly updated to newly observed snapshots while predicting the next snapshot. To predict links

Table 6: Test inference time comparison for snapshot and event based methods on CTDG datasets, results reported from 5 runs. Top three models are coloured by **First**, Second, *Third*.

	Method	`tgbl-wiki`	`tgbl-review`	Reddit
event	TGN [15]	39.24	1137.69	286.79
	DyGFormer [28]	7196.52	26477.51	OOT
	NAT [17]	340.51	8925.21	1159.19
	GraphMixer [37]	1655.44	4167.63	7166.24
	EdgeBank$_\infty$ [16]	20.06	**140.25**	**26.91**
	EdgeBank$_{\text{tw}}$ [16]	20.67	143.49	27.08
snapshot	HTGN (UTG) [12]	28.96	718.17	117.05
	GCLSTM (UTG) [13]	20.54	436.30	82.88
	EGCNo (UTG) [14]	*20.15*	433.23	84.85
	GCN (UTG) [1]	**18.25**	*384.51*	*78.78*

in $\mathbf{G}_{t+1}$, first the observed snapshot $\mathbf{G}_{t-1}$ are split into a training set and a validation set. The model is trained on $\mathbf{G}_{t-1}^{train}$ while using $\mathbf{G}_{t-1}^{val}$ for early stopping. Lastly, the trained model receives $\mathbf{G}_t$ and predicts for $\mathbf{G}_{t+1}$.

Figure 4 illustrates the difference between these three settings. In this work, we focus on the streaming setting as it closely resembles the common use case where even after a model is trained, it is expected to incorporate new information from the data stream for accurate predictions.

C Temporal Graph Learning Methods

C.1 Snapshot-based Methods

snapshot-based methods receives a sequence of graph snapshots as input, representing the temporal graph at specific time intervals (hours, days, etc.). Therefore, DTDG methods are designed to process entire snapshot at once (often with a graph learning model) and then utilize mechanisms to learn temporal dependencies between snapshots. Example methods are as follows:

- **HTGN.** Many DTDG methods focus on learning structural and temporal dependencies in an Euclidean space thus omitting the complex and hierarchical properties which arises in real world networks. To address this, Yang et al. [12] proposed a Hyperbolic Temporal Graph Network (HTGN) which utilizes the exponential capacity and hierarchical awareness of hyperbolic geometric. More specifically, HTGN incorporates hyperbolic graph neural network and hyperbolic gated recurrent neural network to capture the structural and temporal dependencies of a temporal graph, implicitly preserving hierarchical information. In addition, the hyperbolic temporal contextual self-attention module is used to attend to historical states while the hyberbolic temporal consistency module ensures model stability and generalization.
- **GCLSTM.** To learn over a sequences of graph snapshots, Chen et al. [13] proposed a novel end-to-end ML model named Graph Convolution Network embedded Long Short-Term Memory (GC-LSTM) for the dynamic link prediction task. In this work, the LSTM act as the main framework to learn temporal dependencies between all snapshots if a temporal graph while GCN is applied on each snapshot to capture the structural dependencies between nodes. Two GCNs are used to learn the hidden state and the cell state for the LSTM and the decoder is a MLP mapping the feature at the current time back to the graph space. The design of GC-LSTM allows it to handle both link additions and link removals.
- **EGCN.** Existing approaches often require the knowledge of a node during the entire time span of a temporal graph while real world networks often changes its node set. To address this challenge, Pareja et al. [14] proposed the EvolveGCN (EGCN) model which captures the dynamic of the graph sequence by using an RNN to update the weight of a GCN. In this way, The RNN regulates the GCN model parameter directly and effectively performing model adaptation. This allows node changes because the learning is performed on the model itself, rather than specific sequence of node embeddings. Note that the GCN parameters are not trained and only computed from the RNN.

Empirically, the model achieves good performance for link prediction, edge classification and node classification on DTDGs.

- **PyG-Temporal**. PyTorch Geometric Temporal (PyG-Temporal) is an open-source Python library which combines state-of-the-art methods for neural spatiotemporal signal processing [22]. Many existing methods such as EGCN, GCLSTM and more are implemented directly in PyG-Temporal for research. PyG-Temporal is designed with a simple and consistent API following existing geometric deep learning library such as Pytorch Geometric [40]. Originally, PyG-Temporal are designed for node level regression tasks on datasets available exclusively within the framework. In this work, we apply PyG-Temporal models for the link prediction tasks on publicly available datasets.

C.2 Event-based Methods

Continuous Time Dynamic Graph (CTDG) methods receive a continuous stream of edges as input and make predictions over any possible timestamps. CTDG methods incorporate newly observed information into its predictions by updating its internal representation of the world. For efficiency, the stream of edges are divided into fixed size batches while predictions are made for each batch sequentially. To incorporate the latest information, edges from each batch becomes available to the model once the predictions are made. Different from DTDG, CTDG has no inherent notion of graph snapshots, models often track internal representations of a node over time and sample temporal neighborhoods surrounding the node of interest for prediction.

- **TGAT.** Xu et al. [41] argued that models for temporal graphs should be able to quickly generate embeddings in an inductive fashion when new nodes are encountered. The key component of the proposed Temporal Graph Attention (TGAT) layer is to combine the self-attention mechanism with a novel functional time encoding technique derived from Bochner's theorem from classical harmonic analysis. In this way, a TGAT layer can efficiently learn from temporal neighborhood features as well as temporal dependencies. The functional time encoding provides a continuous functional mapping from the time domain to a vector space. The hidden vector of time then replaces positional encoding used in the self-attention mechanism.
- **TGN.** Rossi et al.. [15] introduce Temporal Graph Network (TGN), a versatile and efficient framework for dynamic graphs, represented as stream of timestamped events. TGN leverage a combination of memory modules and graph-based operators to improve computational efficiency. Essentially, TGN is a framework that subsumes several previous models as specific instances. When making predictions for a new batch, TGN first update the memory with messages coming from previous batches to allow the model to incorporate novel information from observed batches.
- **CAWN.** Causal Anonymous Walks (CAWs) [26] are proposed for representing temporal networks inductively to learn the laws governing the link evolution on networks such as the triadic closure law. CAWs, derived from temporal random walks, act as automatic retrievals of temporal network motifs, avoiding the need for their manual selection and counting. An anonymization strategy was also proposed to replace node identities with hitting counts from sampled walks, maintaining inductiveness and motif correlation. CAWN is a neural network model proposed to encode CAWs, paired with a CAW sampling strategy that ensures constant memory and time costs for online training and inference.
- **TCL.** TCL [42] effectively learns dynamic node representations by capturing both temporal and topological information. It features three main components: a graph-topology-aware transformer adapted from the vanilla Transformer, a two-stream encoder that independently extracts temporal neighborhood representations of interacting nodes and models their interdependencies using a co-attentional transformer, and an optimization strategy inspired by contrastive learning. This strategy maximizes mutual information between predictive representations of future interaction nodes, enhancing robustness to noise.
- **NAT.** In modeling temporal networks, the neighborhood of nodes provides essential structural information for interaction prediction. It is often challenging to extract this information efficiently. Luo et al. [17] propose the Neighborhood-Aware Temporal (NAT) network model that introduces a dictionary-type neighborhood representation for each node. NAT records a down-sampled set of neighboring nodes as keys, enabling fast construction of structural features for joint neighborhoods. A specialized data structure called N-cache is designed to facilitate parallel access and updates on GPUs.

- **EdgeBank.** EdgeBank [16] is a non-learnable heuristic baseline which simply memorizes previously observed edges. The surprisingly strong performance of EdgeBank in existing evaluation inspired the authors to also propose novel, more challenging and realistic evaluation protocals for dynamic link prediction.
- **DyGFormer DyGFormer.** Yu et al. [28] introduces a transformer-based architecture for dynamic graph learning. DyGFormer focuses on learning from nodes historical first-hop interactions and employs a neighbor co-occurrence encoding scheme to capture correlations between source and destination nodes through their historical sequences. A patching technique was also proposed to divide each sequence into patches for the transformer, enabling effective utilization of longer histories. *DyGLib* was also presented as a library for standardizing training pipelines, extensible coding interfaces, and thorough evaluation protocols to ensure reproducible dynamic graph learning research.

D Computing Resources

For our experiments, we utilized one of the following GPUs. The first option was NVIDIA A100 GPUs (40GB memory) paired with 4 CPU nodes. These nodes featured CPUs such as the AMD Rome 7532 @ 2.40 GHz with 256MB cache L3, AMD Rome 7502 @2.50 GHz with 128MB cache L3, or AMD Milan 7413 @ 2.65 GHz with 128MB cache L3, each equipped with 100GB memory. The second option was using NVIDIA V100SXM2 GPUs (16GB memory) alongside 4 CPU nodes, which housed Intel Gold 6148 Skylake CPUs @ 2.4 GHz, each with 100GB memory. Our last choice was to run experiments using NVIDIA P100 Pascal GPUs (12GB HBM2 memory) with 4 CPU nodes from Intel E5-2683 v4 Broadwell @ 2.1GHz with 100GB memory. Each experiment had a five-day time limit and was repeated five times, with results reported as averages and standard deviations. Notably, aside from methods adopted from the PyTorch Geometric library, several other models (assessed using their original source code or the DyGLib repository) encountered out-of-memory or out-of-time errors when applied to larger datasets.

E Model Configurations

For all methods and datasets, we employed the Adam optimizer with a two different learning rates namely 0.001 and 0.0002, and the configuration with the higher average performance was selected for reporting the results. Each experiment was repeated five times and the average and standard deviations were reported.

The train, validation, and test splits for `tgbl-wiki` and `tgbl-review` are provided by the `TGB` benchmark. For other datasets (namely, UCI, Enron, Contacts, Social Evo., MOOC, and Reddit), we used a chronological split of the data with 70%, 15%, and 15% for the training, validation, and test set, respectively, which is inline with previous studies [15–17, 25]. We set the batch size equal to 64 for NAT, and for all other models (i.e., TGN, DyGFormer, GraphMixer, EdgeBank, HTGN, GCLSTM, EGCNo, and GCN) the batch size was 200. For the experiments on CTDGs, we set the number of epoch equal to 40 and implemented an early stopping approach with a patience of 20 epochs and tolerance of 10^{-5}. For the experiments on DTDGs, the number of epochs was set to 200 with a similar early stopping approach. Dropout was set to 0.1. We set the number of attention heads equal to 2 for the models with an attention module, and node embedding size was fixed at 100. For TGN, the time embedding size was 100 and the memory dimension was specified as 172, with a message dimension of 100. For NAT, we set the `bias=1e-5`, and *replacement probability=0.7*. All other parameters were set according to the suggested values by Luo and Li [17]. The special hyperparameters of the DyGFormer and GraphMixer are set according to the recommendations presented by Yu et al. [28].