Looking into the past: exploring recurring patterns in time dependencies

Kiril M. Bikov
Department of Computer Science
and Technology
University of Cambridge
kmb85@cam.ac.uk

Riccardo Conci
Department of Computer Science
and Technology
University of Cambridge
rc667@cam.ac.uk

Abstract

Temporal graph methods can handle data that evolves over time, but they often focus only on the most recent information that is closer to the time of a prediction through a temporal neighbourhood defined by a context window. This creates a recency bias and poses the question of how well temporal graphs can utilize recurrent patterns spanning over longer periods of time. To address this research topic, we quantify the recurrent patterns across the common TGL benchmark datasets collected by Temporal Graph Benchmark [7]. Based on the recurrent interactions in each of the TGB datasets, we train and evaluate DyGFormer [29] and GraphMixer [2] on recurrent interactions that occur within the temporal context window. Finally, we create a synthetic dataset with recurrent interactions that recur only once within the context window to further assess memory-based and memory-less temporal graph models. Our extensive results provide novel insights into the ability of temporal graph models to handle recurring patterns over time.

1 Introduction

Dynamic graphs are a valuable representation of many real-world scenarios, including recommendation systems [18], fraud detection [23], biological networks [12], and more [8]. In these networks, nodes, edges, weights, and attributes can be added, deleted, or adjusted over time, making it a challenging problem. Temporal Graph Learning (TGL) [8] [21] [5] has risen as a popular and successful methodology to learn representations over these dynamic graphs for downstream tasks, such as node classification and link prediction [17].

Temporal graph networks extend the notion of a neighbourhood back in time to a temporal neighbourhood, which defines how many previous interactions are taken into account when computing a node's embedding. This context window leads to a recency bias: only the most recent interactions are used to update a node's embedding. Recurrent interactions that span outside this context window are often neglected. Therefore, we aim to explore this recency bias through extensive evaluation of recurrent patterns.

2 Background

Recurrency. Strictly defined, an event that happens exactly every n timesteps is recurrent [13]. This definition can be loosened by adding a percentage noise, such as $n + / - \epsilon$, where ϵ is a percentage of n. However, real-world dynamic tasks such as social networks rarely observe such a strict definition of recurrence. Instead, in practical applications, recurrency is often specified as an event that occurs

within specific intervals (every day, week, month, etc.). We will refer to these types of recurrency as 'count' based and 'interval' based.

Temporal neighbourhoods. Temporal graph models make use of two methods to integrate previous graph interactions into useful node representations: the temporal neighbourhood [1], which defines the interactions back in time, and memory modules, which use recurrent neural networks (RNNs) [20] to update a node's embedding. The temporal neighbourhood is often denoted as : $\mathbf{s}_i(t_0) = \mathbf{x}_i^{\text{node}} + \text{MSP}\left\{\mathbf{x}_j^{\text{node}} \mid v_j \in \mathcal{N}(v_i; t_0 - T, t_0)\right\}$, where $\mathbf{s}_i$ is the encoded node feature at time t_0, $\mathbf{x}_i$ is the original node features, and $\mathcal{N}(v_i; t_0 - T, t_0)$ is the temporal neighbourhood, where T is a a hyperparameter [2]. MSP stands for a message-passing framework [6] [9]. This formulation is nearly ubiquitous across models and defines the recency bias: only nodes within the temporal neighbourhood are used to update embeddings. Figure 1 shows a simplified version of temporal neighbourhood. Node u red occurs every second-time step, while node u $green$ occurs every 5th-time step. With a context window of 4 steps, the model is likely to successfully predict the next red interaction but not the $green$ one.

Memory. Memory-based models keep an updated memory unit for each node. This allows longer-term interactions to be taken into account even if they do not appear directly in the temporal neighbourhood. However, having an RNN-based memory risks challenging gradients when training and an over squashing of information over time. Memory-based models include Joint Dynamic User-Item Embeddings (JODIE) [10], Temporal Graph Networks [18], and APAN [30]. Memory-less models instead include TGAT [27], GraphMixer [2] and DyGFormer [29].

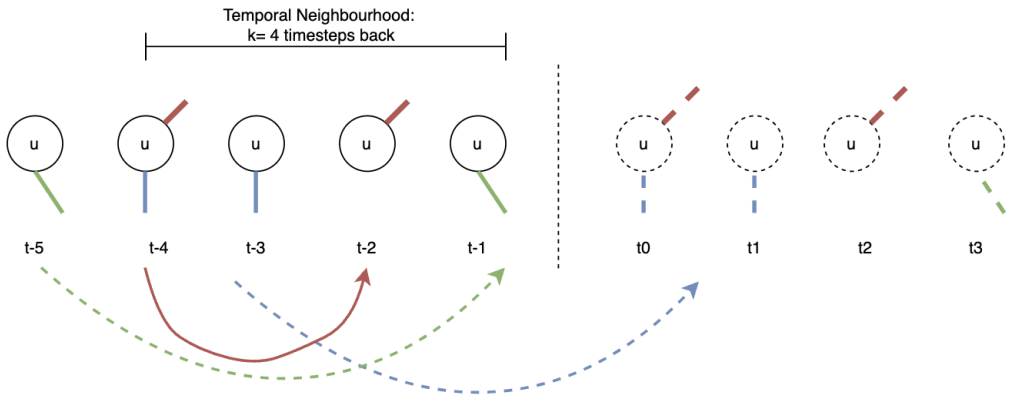


Figure 1: Temporal neighbourhoods - t0 is current time step, negative numbers are going back in time. Dotted are future interactions. Each colour represents a separate destination node for the source node u. Each curved line indicates the recurrent period for each node-node interaction.

3 Methods

3.1 Datasets

TGB datasets. To evaluate TGL models on their ability to detect and utilize recurrency, we use the Temporal Graph Benchmark (TGB) [7]. From TGB, we explore the tgbl-wiki-v2 [10], tgbl-review-v2 [15] tgbl-coin [19], tgbl-tgbl-comment [14], and tgbl-flight [22] datasets. Each of the datasets has different types and frequencies of recurrent patterns. Furthermore, depending on the dataset, those patterns can span over shorter or longer periods of time. This inherent difference in recurrence between the datasets allows us to quantify various models' abilities to predict events that reoccur within the temporal neighbourhood context window.

Synthetic dataset. We created a toy synthetic dataset to precisely assess how models deal with events that reoccur with a period longer than their temporal context window, targeting the recency bias. The dataset is built to test a context window of 8 prior interactions. It has 8 source and 8 destination nodes, which interact regularly every 8 timesteps. This pattern repeats for a total of 5120 total interactions,

as shown in figure 2. The synthetic setup forces each node to only ever see 8 distinct interactions in its recent past, not allowing it to pick up any recurrence within its temporal neighbourhood.

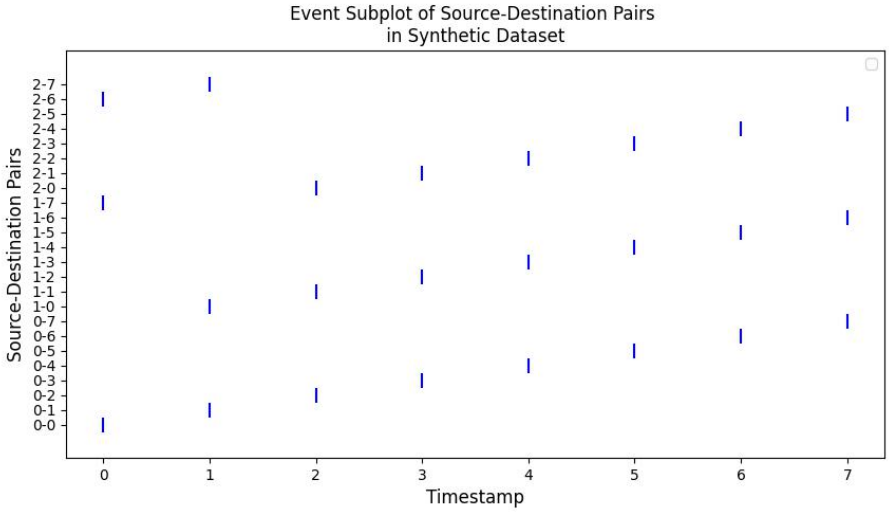


Figure 2: Subset of the synthetic dataset. Each node-node interactions repeats every 8 time steps.

3.2 Models

Recurrency experiments. To assess how temporal graph models perform on recurrent data that is repeated within their context window, we trained and tested DyGFormer [29] and GraphMixer [3] on recurrent subsets of the TGB datasets. DyGFormer [29] has a Transformer-based dynamic graph learning architecture. The model utilizes a neighbour co-occurrence encoding scheme, splits each node's sequence into multiple patches and feeds them to Transformer [26]. The architecture of GraphMixer consists of three components: a link-encoder, a node-encoder, and an MLP-based link classifier [3]. The link-encoder summarizes the temporal link information associated with each node [24]. The node-encoder is designed to capture the node identity and node feature information via neighbour mean-pooling, and the link classifier determines whether a link exists at a given time [3].

Each model is run based on the optimal hyperparameters suggested by the DyGLib_TGB code base [11]. Further information on how we trained the models can be found in the Appendix 17.

Recency bias experiments. We extend the evaluation on the synthetic dataset to include TGAT [27], JODIE [10], Temporal Graph Networks [18], and DyRep [25]. Experiments involved training these models on a context window of 8 and then of 32 to see if increasing would improve prediction. Each model was trained for 20 epochs with early stopping, a batch size of 200 (other than TGN with a batch size of 1), and a standard embedding size of 100. Each experiment was repeated three times with different random seeds to give an average Mean Reciprocal Rank (MRR) score with a distribution.

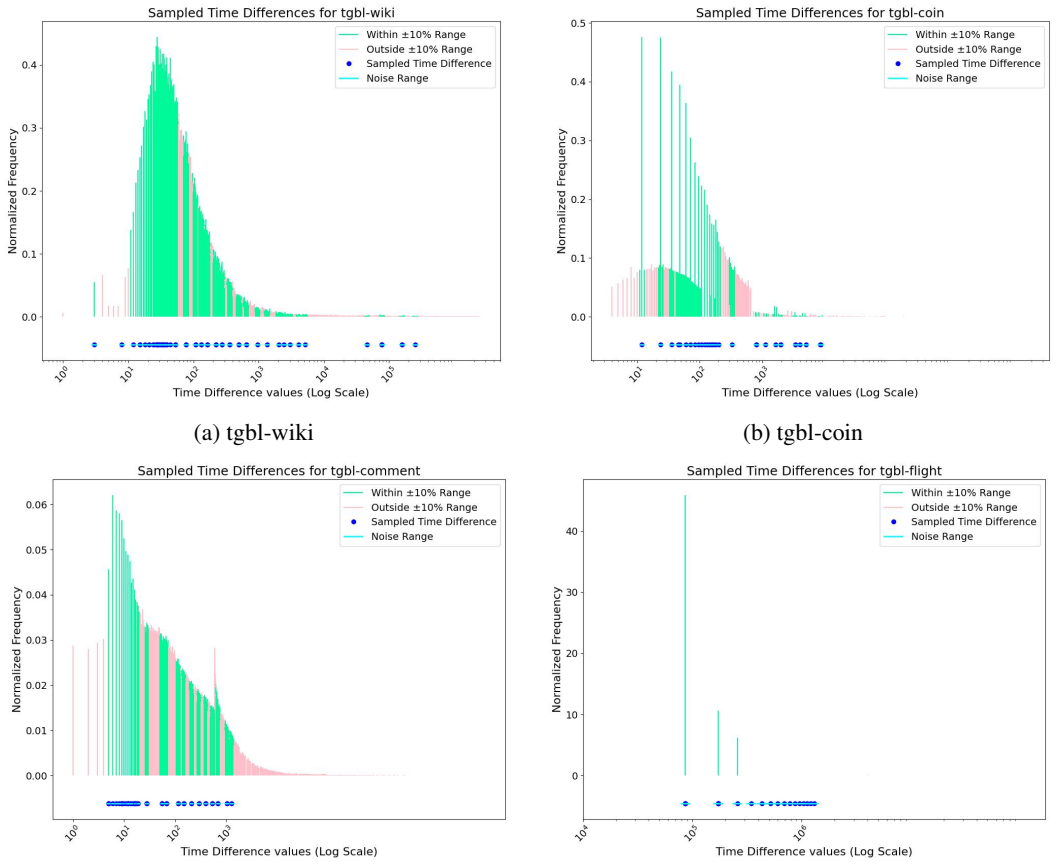
3.3 Recurrency in datasets

Previous work [16] introduced the Temporal Edge Traffic (TET) plots, describing the percentage of edges that are only part of the training set, those that cross between training and validation, and those only found in the test set. We begin by computing the percentage of each of these data subsections that shows recurrency. This quantification gives an indication of how heavily models using that dataset are trained and tested on recurrent patterns.

To precisely quantify the recurrence of the datasets regardless of the period of recurrent events, we propose an algorithm in the Appendix 17. We begin by finding the time differences (TDs) in the dataset and ranking them by count. We then pick the 15 most common TDs, and to that, we sample

another 5-30 as required to have good coverage of the dataset. The sampling is weighted by the count. Finally, each TD is given a 10% noise around it, which allows for a slightly looser definition of count-based recurrency.

The effectiveness of this method is shown in figure 3 and in table 3. Each sub-figure shows the time differences covered by the sampling process. The height of the lines indicates the count associated with each TD. The colour indicates if it is covered. The blue dots indicate the sampled TDs, with the cyan noise around them. As shown, tgbl-flight is highly recurrent, requiring only 3-4 TD samples to account for over 70% of interactions. tgbl-comment and tgbl-coin show medium levels of recurrency, requiring more samples to adequately cover the dataset. tgbl-wiki has no dataset-wide recurrency but rather has a few very recurrent and frequent individual source-destination pairs, which drive the smoothness of the TDs. Finally, tgbl-review has very little recurrency that occurs with a time difference greater than 0. Inherently, tgbl-review is unlikely to be recurrent, as it includes Amazon reviews of electrical products by users. In its results on Table 1, it does not pick up any recurrence with a 10% noise. Because of this, we place the tgbl-review plot in Appendix 6.



(a) tgbl-wiki

(b) tgbl-coin

(c) tgbl-comment

(d) tgbl-flight

Figure 3: Sampling Time Differences to quantify recurrency. X-axis: time differences in the dataset increasing in value (log scale). Y-axis: normalised count for each time difference in the dataset (summing to 100). Pink vs green: whether the dataset time difference is taken into account by the sampled time difference +/- noise value. Blue dots: the sampled time difference frequencies. Cyan lines: the noise window around each sampled time frequency.

3.4 Modelling recurrent interactions within the context window

Our experiments aim to explore recurrent patterns in the Temporal Graph Benchmark [7] datasets and measure how well Dynamic Graph Learning Models [28] can learn from those patterns. We focus our evaluation on DyGFormer and GraphMixer [28] described in section 3.4.

As discussed in Section 2, models provide different methods for integrating historical data to compute embeddings. In general, models use a fixed number of interactions for each source node back in time as part of their context window. This number can be adjusted. Given the need to evaluate these models on the looser count-based definition of recurrence, the first step is to measure the average datetime that corresponds to the various context window lengths that are used by the models on our datasets.

Figure 18 shows the average time frame based on the sequence length. As expected, a higher context window leads to a larger time frame. It is worth noting that these time frames are averages with a large spread of results and, therefore, are biased by outliers. This impact is seen in the tgbl-flight dataset, which shows the average time frame decreasing with increasing context windows. From figure 18, the time frame of tgbl-wiki ranges between a few weeks, whereas for tgbl-flight, tgbl-comment and tgbl-review, the context window can span over years.

Now that we have data on the average time frame that the models explore depending on the sequence length, we can evaluate the recurrent pattern in this time frame. We begin by processing the datasets so that only recurrent pairs with the given frequency that start in the training dataset and continue through the validation dataset are included. The recurrent frequency is based on the loose version of recurrency - it looks for a given count of occurring interactions within each repeating time frame. For example, the historical context window of 32 interactions for the tgbl-wiki dataset averages around 15 days. To process the dataset, we look at all interactions in the last 15 days of the training data and all the validation data. From this data and for a given recurrence frequency of once a week, we only obtain pairs of nodes that interact at least one time every consecutive week. Finally, we use the training split to train models from scratch on the recurrent interactions and test them on the continuing recurrent interactions in the evaluation subset.

Sequence Length	tgbl-wiki	tgbl-flight	tgbl-comment	tgbl-review
8	13d 10h 34m	1y 78d	355d 20h 2m	174d 4h 19m
16	13d 23h	1y 112d	1y 112d 15h	250d 21h 21m
32	15d 6h 29m	1y 91d	1y 261d 6h	1y 72d 17h
128	17d 19h 57m	1y 50d 12h	2y 27d 21h	2y 350d 12h
512	19d 23h 10m	1y 207d	2y 151d 17h	3y 170d 6h
1024	26d 14h 51m	1y 195d 12h	3y 17d 2h	5y 260d 4h
4096	-	2y 64d 12h	3y 287d 11h	6y 248d 18h

Figure 4: Average Time Frame(Context Window) based on sequence length

To better understand the recurrency in the tgbl-wiki, tgbl-comment, tgbl-flight and tgbl-review datasets, we also display the number of interactions and nodes in the training and validation splits, as shown in figures 11 13 14 12. The ratio between the number of nodes and recurrent interactions has an important effect on the performance of the models discussed in section 4.2.

4 Results

4.1 Recurrency across Datasets

Temporal Edge Traffic (TET) plots [16] describe the percentage of interactions that are only part of the training set, those that cross between training and validation, and those only found in the test set. As a continuation of this work, Table 1 shows the percentage of these interactions in each subset and the percentage of those that are recurrent. We do not include tgbl-review as only around 400 source-destination events with a time difference of over 0 seconds were found, of which only a few had at least four interactions.

Dataset	% Recurrency overall	% Train	% Recurrent of Train	% Train-Val	% Recurrent of Train-Val	% Val +Test	% Recurrent of Val+Test
tgbl-wiki	0.66	7.46	1.13	1.57	0.44	2.54	0.92
tgbl-coin	0.72	11.1	0.68	2.59	0.81	3.16	0.48
tgbl-comment	0.034	58.9	0.009	0.69	0.036	20.5	0.005
tgbl-tgbl-flight	16.4	1.91	2.93	0.92	4.80	0.60	2.45

Table 1: Recurrency across datasets

4.2 Recurrent interactions within the context window

In this section, we show our results on training and evaluating GraphMixer [3] and DyGFormer [29] on the tgbl-wiki, tgbl-review, tgbl-comment and tgbl-flight datasets. The training has been performed for one run over 16 and 32 epochs.

Figure 5 shows that on recurrent events in the tgbl-wiki dataset, both GraphMixer and DyGFormer perform very well with over 0.85 validation MRR. With a frequency of once a week, GraphMixer achieves better results given a lower sequence length, whereas DyGFormer excels with a bigger sequence length of 128. As the frequency increases, both models perform remarkably well with a validation MRR close to 1. Based on the results from figure 5, we conclude that on recurrent data with many nodes and interactions that happen close to each other in time(within weeks), both GraphMixer and DyGFormer perform exceptionally well with validation MRR between 0.83 and almost a 1. Nevertheless, DyGFormer does have a slight advantage over GraphMixer, as it produces consistent validation MRR over 0.9 for any combination of sequence length and frequency.

Seq	Frequency	GraphMixer 16eps MRR	GraphMixer 32eps MRR	DyGFormer 16eps MRR	DyGFormer 32eps MRR
32	once	0.9962	0.9962	0.9403	0.9539
32	twice	0.8292	0.8609	0.9813	0.9752
32	3-times	0.9781	0.8911	0.9831	0.9852
128	once	0.8867	0.8833	0.9808	0.9788
128	twice	0.9898	0.9908	0.9969	0.9969
128	3-times	0.9962	0.9932	0.9962	0.9962

Figure 5: Recurrency evaluation on the tgbl-wiki dataset, Seq-Sequence Length, eps - number of epochs, MRR - Validation Mean reciprocal rank

Next, we evaluate the models on the tgbl-review dataset, which almost completely lacks any recurrent interactions. Based on the very scarce recurrent data, both DyGFormer and GraphMixer produce relatively low validation MRR of around 0.5, as shown in figure 6. Therefore, we argue that even when the data is recurrent, there is a need for a sufficiently large number of training and validation interactions for the models to produce good predictions.

Seq	Frequency	GraphMixer 16eps MRR	GraphMixer 32eps MRR	DyGFormer 16eps MRR	DyGFormer 32eps MRR
32	twice in 1 year	0.5050	0.0148	0.5098	0.5098
128	twice in 2 years	0.5892	0.1569	0.5464	0.5464

Figure 6: Recurrency evaluation on the tgbl-review dataset

On the tgbl-comment dataset, figure 7 shows that GraphMixer varies from 0.78 to 0.99 Validation MRR, whereas DyGFormer ranges between 0.55 and almost a 1. A general observation is that GraphMixer achieves more stable performance across different sequence lengths and frequencies. In comparison, DyGFormer's accuracy varies significantly more, achieving almost perfect results for a small sequence length of 32 but struggling with a higher length of 128. Based on the results in figure 7, we conclude that for recurrent data with moderate interactions of occasional frequency and fewer nodes, GraphMixer provides better, more stable performance than DyGFormer.

Seq	Frequency	GraphMixer 16eps MRR	GraphMixer 32eps MRR	DyGFormer 16eps MRR	DyGFormer 32eps MRR
32	once a month	0.9921	0.9913	0.6579	0.9989
32	3-times a month	0.9932	0.99324	0.9992	0.9991
128	once a month	0.9971	0.9973	0.7568	0.7572
128	3-times a month	0.7729	0.7889	0.5464	0.5464

Figure 7: Recurrency evaluation on the tgbl-comment dataset

Lastly, on the tgbl-flight dataset in figure 8, DyGFormer substantially outperforms GraphMixer. Whereas GraphMixer struggles with validation MRR ranging between 0.167 and 0.55, DyGFormer produces a high validation MRR between 0.73 and almost 1. The significant difference in performance shows the advantage of DyGFormer on recurrent events that are very frequent over a long period of time. When presented with recurrent data that has a large number of interactions occurring often, DyGFormer achieves much better performance compared to GraphMixer.

Seq	Frequency	GraphMixer 16eps MRR	GraphMixer 32eps MRR	DyGFormer 16eps MRR	DyGFormer 32eps MRR
32	twice a day	0.1698	0.4605	0.7278	0.7328
32	3-times a day	0.5399	0.5238	0.9691	0.9791
128	twice a day	0.1203	0.5525	0.9805	0.9807
128	3-times a day	0.5461	0.5459	0.9811	0.9812

Figure 8: Recurrency evaluation on the tgbl-flight dataset

4.3 Assessing the recency bias on synthetic data

Table 2 shows the results of the training and testing on the synthetic data. The models are ranked by results, with DyGFormer leading across both the context window of 8 and 32 and TGN able to perform with a context window of 32. DyGFormer is significantly better than any other model, especially with a context window of 8, and has nearly perfect performance with a window of 32. This is likely due to the correlation module of DyGFormer, which allows it to indirectly infer the recurrence pattern between nodes. TGN approaches the performance of DyGFormer as the context window increases to 32. The remaining models produce poor predictions regardless of the context window. The results also need to be considered in the context of TGN and JODIE, both being memory-based models, while DyGFormer is not. Instead, DyGFormer has the correlations and the transformer architecture to manage recurrency. Further plots of the predictions on the validation dataset by TGN with a context window of 32 can be found in the Appendix 17.

Model	Validation MRR Window = 8	Test MRR Window = 8	Validation MRR Window = 32	Test MRR Window = 32
DyGFormer	0.896 ± 0.04	0.896 ± 0.04	0.978 ± 0.012	0.977 ± 0.012
TGN	0.470 ± 0.006	0.470 ± 0.006	0.7626 ± 0.001	0.7626 ± 0.001
JODIE	0.357 ± 0.05	0.357 ± 0.05	0.352 ± 0.005	0.352 ± 0.005
GraphMixer	0.356 ± 0.16	0.356 ± 0.16	0.303 ± 0.043	0.303 ± 0.043
DyRep	0.34 ± 0.00	0.34 ± 0.00	0.34 ± 0.00	0.34 ± 0.00
TGAT	0.312 ± 0.09	0.312 ± 0.09	0.32 ± 0.07	0.32 ± 0.07
EdgeBank	0.2222 ± 0.0	0.2222 ± 0.0	0.2222 ± 0.0	0.2222 ± 0.0

Table 2: Results on Synthetic dataset

5 Discussion

We first explored how well TGL models perform when trained and evaluated on recurrent data that span within their context window 4.2. Based on the results from figures 5 6 7 14, we argue that

both GraphMixer and DyGFormer can provide very good performance when trained on sufficiently large data in terms of number of nodes and interactions. Depending on the ratio between nodes and interactions, the sequence length and the frequency of interactions, we observed cases when GraphMixer outperforms DyGFormer and such where DyGFormer is notably better. A general observation of our finding is that both GraphMixer and DyGFormer are able to handle recurrent interactions within their context window very well when presented with sufficiently large data.

Our further experiments on a synthetically created data provided new insights on the ability of the models to handle recurrency outside their context window 2. We observed that the majority of models such as JODIE, GraphMixer and TGAT are unable to learn and predict the simple repeating pattern in the synthetic data. The only model that produced consistent high results on the synthetic data is DyGFormer, with TGN also achieving above average results. Based on those results, we conclude that learning and predicting recurrent patterns outside the context window is a significant challenge for most models that needs to be addressed.

One potential solution to this challenge is to increase the context window on an edge basis. To visualise this proposed solution, figure 9 shows an example of an enhanced context window, where for every initial node in the temporal neighbourhood, a further $n = 3$ previous interactions for each source-destination pair are also appended to the model. n does not need to be fixed either and can be learned based on the dataset and temporal neighbourhood structure. Adding this enhanced window is a potential way to maintain the ability to deal with low frequency recurrent interactions without a memory module that may ultimately become oversquashed [4].

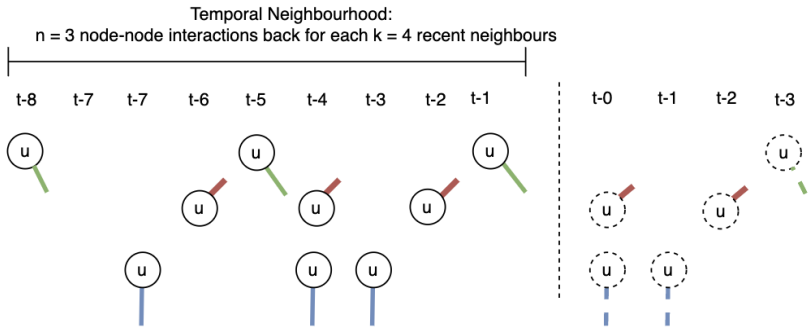


Figure 9: Enhanced temporal neighbourhood, where for each initial neighbourhood node, the previous n=3 interactions are also taken

6 Conclusion

We provided novel results on how well temporal graph models handle recurrent patterns in data. First, we analysed and visualised existing recurrency in the Temporal Graph Benchmark (TGB) datasets [7]. We found that each of the TGB datasets has different levels of recurrent interactions with varying periods of recurrence. Next, we trained and evaluated DyGFormer and GraphMixer on processed subsets of the TGB datasets that only contained interval-based recurrent interactions. Our results showed that both models perform very well on highly recurrent data that is present in the context window of the models. We observed that the choice between DyGFormer and GraphMixer for optimal prediction accuracy depends on the exact dataset, number of interactions, nodes and frequency.

To assess the recency bias in temporal graph models, we created a synthetic dataset. Our evaluation of the models on this dataset showed that most models, including JODIE, GraphMixer, DyRep and TGAT, achieve poor performance and are not able to succeed on this toy task. DyGFormer was the only model that managed to overcome the recency bias and provide excellent results on the synthetic dataset with a larger context window. Our results show that handling recurrent patterns outside the context window is a significant challenge for most temporal graph models. We briefly propose and discuss how this challenge can be addressed by increasing the context window on an edge basis. Future work can build upon our proposed solution in allowing temporal graph models to overcome recency bias in recurrent data.

References

[1] Michael Bronstein. Temporal graph networks.

[2] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, H Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks? *Int Conf Learn Represent*, abs/2302.11636, February 2023.

[3] 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?

[4] Francesco Di Giovanni, T. Konstantin Rusch, Michael M. Bronstein, Andreea Deac, Marc Lackenby, Siddhartha Mishra, and Petar Veličković. How does over-squashing affect the power of GNNs?

[5] Jianfei Gao and Bruno Ribeiro. On the equivalence between temporal and static equivariant graph representations. In *Proceedings of the 39th International Conference on Machine Learning*, pages 7052–7076. PMLR. ISSN: 2640-3498.

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1263–1272. JMLR.org.

[7] 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.

[8] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey.

[9] Kacper Kubara. Introduction to message passing neural networks.

[10] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. August 2019.

[11] Yu Le. DyGLib_TGB: An empirical evaluation of temporal graph benchmark.

[12] Paola Lecca and Michela Lecca. Graph embedding and geometric deep learning relevance to network biology and structural chemistry. *Front Artif Intell*, 6:1256352, November 2023.

[13] Chiara Mocenni, Angelo Facchini, and Antonio Vicino. Comparison of recurrence quantification methods for the analysis of temporal and spatial chaos. *Math. Comput. Model.*, 53(7):1535–1545, April 2011.

[14] 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*, WWW '22, pages 677–685, New York, NY, USA, August 2022. Association for Computing Machinery.

[15] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using Distantly-Labeled reviews and Fine-Grained aspects. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics.

[16] Farimah Poursafaei. DGB: Dynamic graph benchmark.

[17] Farimah Poursafaei, Andy Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction.

[18] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. June 2020.

[19] Kiarash Shamsi, Friedhelm Victor, Murat Kantarcioglu, Y Gel, and C Akcora. Chartalist: Labeled graph datasets for UTXO and account-based blockchains. *Adv. Neural Inf. Process. Syst.*, 2022.

[20] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long Short-Term memory (LSTM) network. August 2018.

[21] Amauri H Souza, Diego Mesquita, Samuel Kaski, and Vikas Garg. Provably expressive temporal graph networks.

[22] Martin Strohmeier, Xavier Olive, Jannis Lübbe, Matthias Schäfer, and Vincent Lenders. Crowdsourced air traffic data from the OpenSky network 2019–2020. *Earth Syst. Sci. Data*, 13(2):357–366, February 2021.

[23] Yue Tian and Guanjun Liu. Transaction fraud detection via Spatial-Temporal-Aware graph transformer. July 2023.

[24] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-mixer: An all-MLP architecture for vision.

[25] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. DyRep: Learning representations over dynamic graphs. September 2018.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need.

[27] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs.

[28] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library.

[29] Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library. November 2023.

[30] 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. March 2022.

Appendix

GitHub Code Repository

Our code for processing the datasets, as well as training and evaluating the models can be found at: GitHub Project Repository(`https://github.com/rickconci/TGL_recurrent`).

Time Difference Sampling

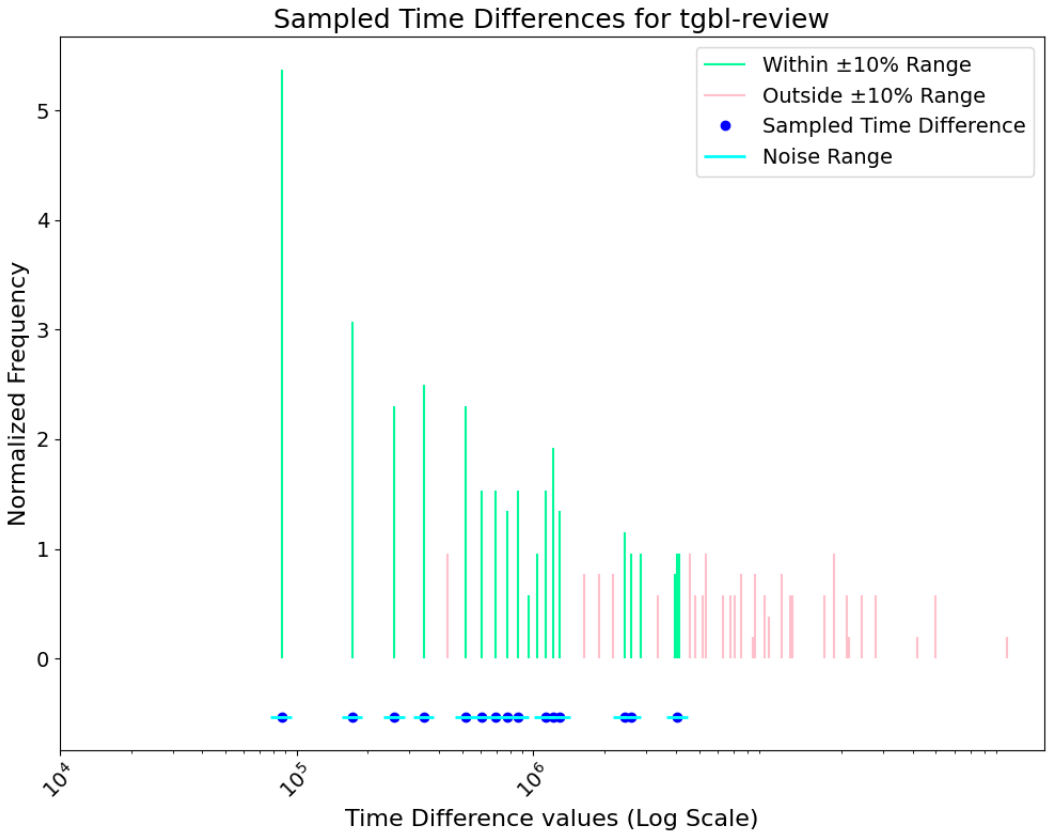


Figure 10: tgbl-review with sampled time differences.

Table 3: Summary of Selected Time Diffs across Datasets.

Dataset	Selected Time Diffs	% Interactions accounted for	Total Interactions	% Time diffs accounted for	Total Time diffs
Wikipedia	42	72	133622	66	31647
Review	12	100	521	100	254
Coin	30	45	17210797	35	2199710
Comment	50	69	4264207	63	1682607
Flight	21	93	29470428	15	3722

Algorithm 1: Sampling recurrent frequencies

```
explored ← top_15_Time_Diff
forbidden ← top_15_Time_Diff_forbidden_ranges
attempts ← 0
while len(explored) < num_total_frequencies and attempts < max_attempts do
    sample ← random.choice(time_diff, time_diff_count, 1)
    if sample ∉ forbidden then
        explored ← Append(explored, sample)
        upper ← sample + (sample × noise_value ÷ 100)
        lower ← sample − (sample × noise_value ÷ 100)
        forbidden_window ← range(lower, upper)
        for val in forbidden_window do
            forbidden ← Append(forbidden, val)
        end
    end
    attempts ← attempts + 1
end
return explored
```

Predicting recurrency within the context window

Seq	Freq	Train Interact	Val Interact	Train Nodes	Val Nodes
32	once a week	28595	6127	295	306
32	twice a week	25344	5430	630	188
32	3-times a week	22148	4746	494	158
128	once a week	14342	3073	312	106
128	twice a week	10575	2266	179	41
128	3-times a week	8562	1834	121	30

Figure 11: Number of Interactions/Nodes for Training/Validation - Wiki Dataset

Seq	Context Window	Train Interact	Val Interact	Train Nodes	Val Nodes
32	twice in 1 year	21	4	8	4
128	twice in 2 year	88	19	35	12

Figure 12: Number of Interactions/Nodes for Training/Validation - Review Dataset

Seq	Context Window	Train Interact	Val Interact	Train Nodes	Val Nodes
32	once a month	8782	3764	35	17
32	3-times a month	3611	1549	10	4
128	once a month	3622	1553	9	7
128	3-times a month	1180	507	2	2

Figure 13: Number of Interactions/Nodes for Training/Validation - Comment Dataset

Seq	Context Window	Train Interact	Val Interact	Train Nodes	Val Nodes
32	twice a day	1048663	224713	43	26
32	3-times a day	744997	159642	33	16
128	twice a day	1031043	220937	54	27
128	3-times a day	726037	155579	35	16

Figure 14: Number of Interactions/Nodes for Training/Validation - Flight Dataset

Model	Seq	Frequency	Precision 16eps	Precision 32eps
DyGFormer	32	once	0.9966	0.9969
DyGFormer	32	twice	0.9966	0.9962
DyGFormer	32	3-times	0.9948	0.9978
DyGFormer	128	once	0.9804	0.9896
DyGFormer	128	twice	0.9664	0.9737
DyGFormer	128	3-times	0.9485	0.9490
GraphMixer	32	once	0.9695	0.9737
GraphMixer	32	twice	0.9697	0.9750
GraphMixer	32	3-times	0.9699	0.8358
GraphMixer	128	once	0.9626	0.9662
GraphMixer	128	twice	0.9389	0.9347
GraphMixer	128	3-times	0.9150	0.9235

Figure 15: Training precision on the recurrent Wikipedia subset, Seq-Sequence Length, Interact - number of interactions, Precision - Average Training Precision, eps - number of epochs

Model	Seq	Frequency	Precision 16eps	Precision 32eps
DyGFormer	32	twice in 1 year	0.8333	0.8810
DyGFormer	128	twice in 2 years	0.8977	0.9261
GraphMixer	32	twice in 1 year	0.7857	0.8810
GraphMixer	128	twice in 2 years	0.9148	0.9205

Figure 16: Training precision on the recurrent Review subset

Model	Seq	Frequency	Precision 16eps	Precision 32eps
DyGFormer	32	once a month	0.9559	0.9665
DyGFormer	32	3-times	0.9234	0.9163
DyGFormer	128	once a month	0.6994	0.7037
DyGFormer	128	3-times a month	0.8977	0.9261
GraphMixer	32	once a month	0.9125	0.9232
GraphMixer	32	3-times a month	0.9106	0.9052
GraphMixer	128	once a month	0.8915	0.8948
GraphMixer	128	3-times a month	0.6847	0.7078

Figure 17: Training precision on the recurrent Comment subset

Model	Seq	Frequency	Precision 16eps	Precision 32eps
DyGFormer	32	twice a day	0.8356	0.8422
DyGFormer	32	3-times a day	0.8329	0.8345
DyGFormer	128	twice a day	0.8791	0.8822
DyGFormer	128	3-times a day	0.8744	0.8794
GraphMixer	32	twice a day	0.5735	0.5686
GraphMixer	32	3-times a day	0.5399	0.5238
GraphMixer	128	twice a day	0.5749	0.5626
GraphMixer	128	3-times a day	0.5461	0.5459

Figure 18: Training precision on the recurrent Flight subset

Synthetic dataset predictions

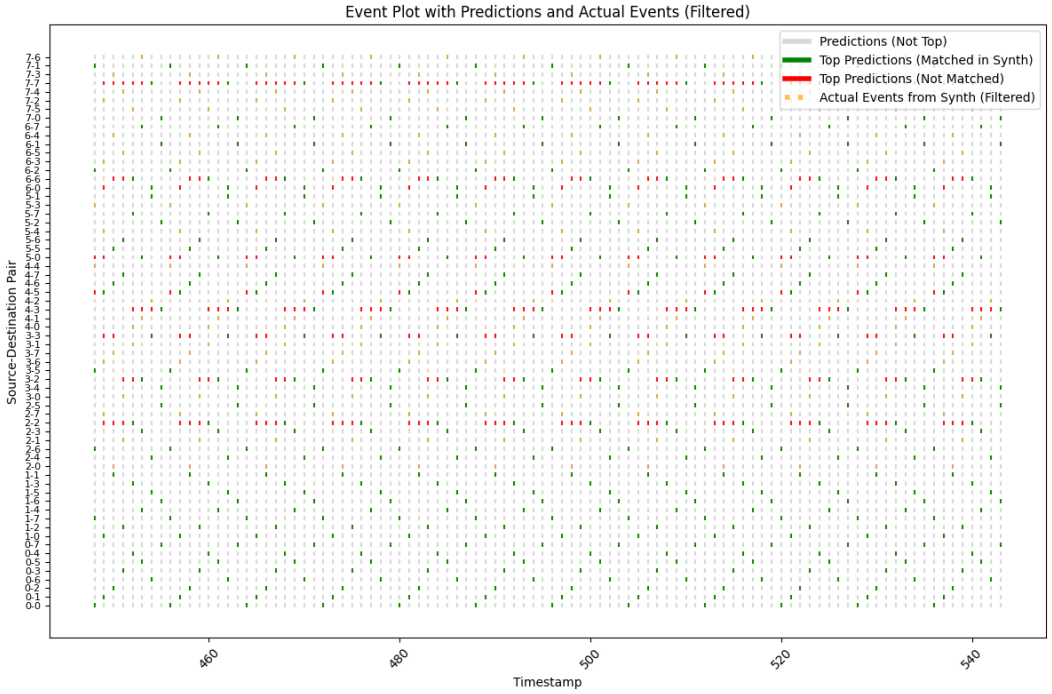


Figure 19: TGN with context window of 32, validation dataset predictions. This shows a performance of around 0.75 MRR.

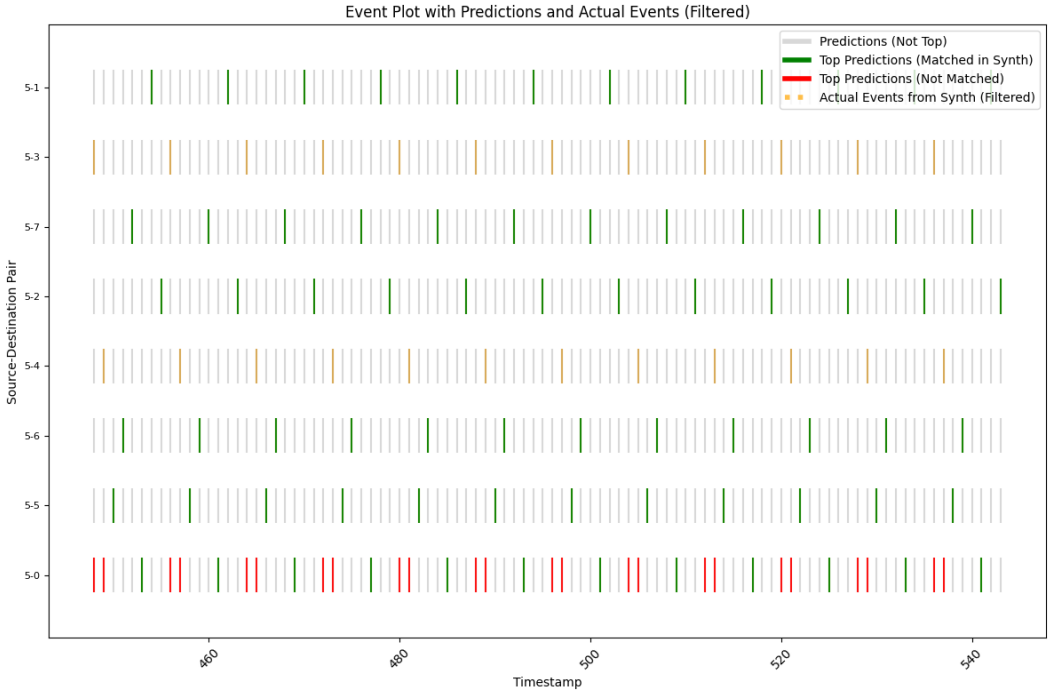


Figure 20: TGN synthetic data predictions for specific source-destination group. Node how the model when unsure will stick to a specific source-destination pair, such as 5-0. As the dataset is symmetric this inbalance is due to the model.