Technical Report: Additional Experiments

Table A1: Comparison of (average) batch size by the traditional and ETC's batching scheme under the same information loss upper bound.

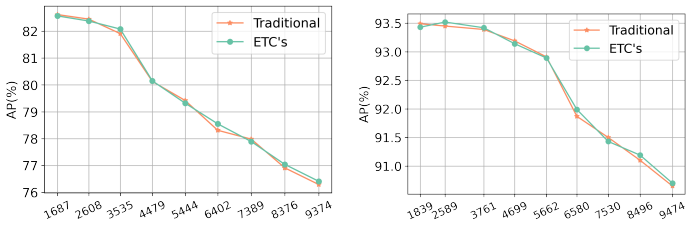
Dataset	Method	Batch Size								
LastFM	Traditional	1000	1500	2000	2500	3000	3500	4000	4500	5000
	ETC's	1180	1704	2210	2714	3221	3718	4226	4731	5240
Wiki-Talk	Traditional	1000	1500	2000	2500	3000	3500	4000	4500	5000
	ETC's	1442	2118	2771	3399	4033	4631	5243	5860	6480

1 Comparison with Traditional Batching Scheme.

We conduct comprehensive case studies that compare ETC's batching scheme with the traditional batching scheme. These case studies cover scenarios with both (1) varied information loss upper bounds and (2) varied (average) batch sizes. In these experiments, we employ TGN as the backbone T-GNN model and present the comparison results using the LastFM and Wiki-Talk datasets. It is worth noting that, in ETC's batching scheme, the input is the information loss threshold rather than a fixed batch size. Consequently, batch sizes may vary across different batches when using ETC's batching scheme. Therefore, we refer to the resultant average batch size in the context of ETC's batching scheme, which takes into account all batches.

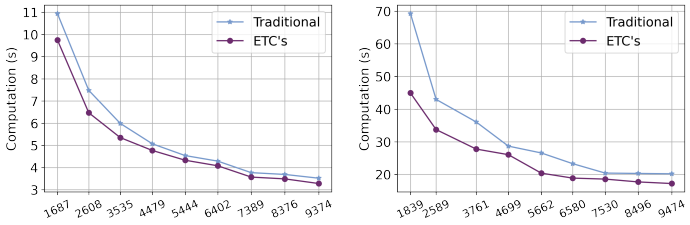
Varied Information Loss Score Upper Bound. In our first case study, we conducted experiments with a batch size range of 1000 to 5000 using a step of 500 for the traditional batching scheme. Subsequently, we calculated the information loss score upper bound for all batches following Eq. 12 in Section 4.1 (the same setup is adopted in the main experiments). We then used this derived information loss score upper bound as the input for ETC's batching scheme to ensure a fair comparison, ensuring that both batching schemes sustaining the same constraint for information loss. Table A1 provides a summary of batch sizes obtained using the traditional batching scheme and the average batch size achieved with ETC's batching scheme. Notably, under the same information loss constraints, ETC's batching scheme resulted in larger average batch sizes. Figures A1 and A2 display the model's predictive performance and per-epoch computation time for both the traditional and ETC's batching schemes. These figures reveal that both batching schemes exhibited similar predictive performance when subjected to the same extent of information loss in their respective batches. However, ETC's batching scheme showcased superior model computation efficiency under the same conditions. This improvement can be attributed to the fact that ETC's batching scheme, with larger batch sizes on average (as shown in Table A1). This, in turn, enhances the parallelized processing of events within the input dynamic graph by GPU.

Varied Batch Size. In our second case study, which focuses on varying batch size, we utilized the average batch sizes obtained from ETC's batching scheme in the first case study (where information loss thresholds are varied) as input for the conventional data batching scheme. Comparing the two batching schemes under the same batch size, we observed that they both demonstrated nearly identical model computation efficiency (Figure A4). However, ETC's batching scheme outperformed the traditional batching scheme in terms of predictive performance for the underlying model ((Figure A3). This is due to the less extent of information loss resulted from ETC's batching scheme compared to the traditional approach (as demonstrated in Table A2).



(a) LastFM **(b) Wiki-Talk**

Figure A1: Average precision comparison between the traditional and ETC's batching scheme under the same information loss score upper bound.



(a) LastFM **(b) Wiki-Talk**

Figure A2: Per-epoch model computation time comparison between the traditional and ETC's batching scheme under the same information loss score upper bound.

In summary, when comparing ETC's batching scheme with the traditional approach, our two case studies reveal that ETC's batching scheme offers faster model computation efficiency while maintaining the same model predictive performance when operating under the same information loss score upper bound. Additionally, when subjected to the same batch size, ETC's batching scheme excels in providing better model predictive performance while maintaining nearly identical model computation efficiency. These empirical findings underscore the superiority of ETC's batching scheme over the conventional method. Furthermore, in response to your suggestion, we have included the (average) batch size information for ETC's batching scheme for the main experiments in the experimental section (training configurations in Section 5.1).

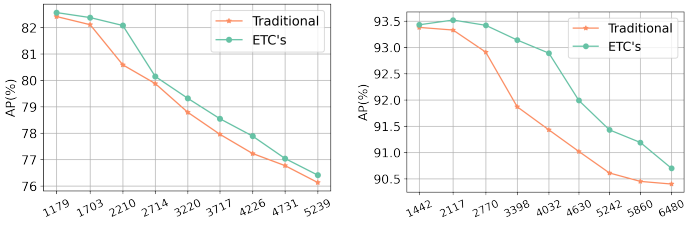
2 Alternative Storage Schemes.

When training existing T-GNNs over large-scale dynamic graphs, all the input data are stored in the large CPU main memory during the training process [9] (all loaded from the secondary storage in an one-shot manner at the beginning of the training process), as the CPU main memory is large enough (generally no less than 256GB) for total input data storage as shown in previous literature [2, 3, 7, 8, 9], while GPU memory is not adequate for extremely large-scale dynamic graphs. ETC follows this default storage scheme for T-GNN training on large-scale dynamic graphs, and the systematic optimizations within ETC are tailored under this scheme as well.

Nevertheless, we explore alternative storage schemes beyond the default one (CPU main memory). In our evaluation, we consider three possible storage schemes during T-GNN training for

Table A2: Comparison of information loss score upper bound by the traditional and ETC's batching scheme under the same (average) batch size.

Dataset	Method	Information Loss Score Upper Bound								
LastFM	Traditional	2004	2978	3933	4888	5862	6827	7830	8837	9850
	ETC's	1687	2608	3535	4479	5444	6402	7389	8376	9374
Wiki-Talk	Traditional	2696	3978	5253	6380	7599	8722	9958	11037	12255
	ETC's	1839	2803	3761	4699	5662	6580	7530	8496	9474



(a) LastFM **(b) Wiki-Talk**

Figure A3: Average precision comparison between the traditional and ETC's batching scheme under the same batch size.

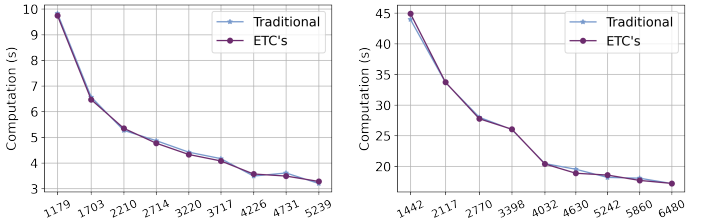
both ETC and the generic T-GNN training framework TGL:

- Storing input data in GPU memory.
- Storing input data in CPU main memory.
- Storing input data in local SSD.

For the SSD storage scheme, input data is divided into partitions and stored in SSD, with subsets of data partitions transferred from SSD to CPU main memory [6]. We selected the LastFM and Wiki-Talk datasets for the evaluation, which enable us to assess the efficiency performance under all three storage schemes within our experimental setup.

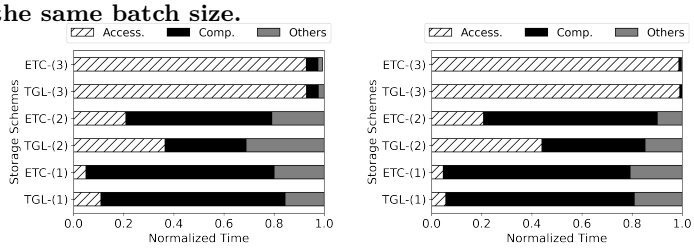
We trained the TGAT model [8] with batch sizes of 1000 and 1500 on these datasets using both ETC and TGL. Figures A5a and A5b provide a breakdown of epoch times for LastFM and Wiki-Talk. Notably, the GPU storage scheme resulted in minimal overhead in terms of data access, accounting for approximately 6% to 10% of the overall training time. However, this storage scheme is often impractical for training T-GNNs at scale due to the limited capacity of GPU memory, as discussed in Section 2.3. With effective CPU-GPU data access optimizations, the data access time still constitutes a relatively small proportion (around 20%) of the overall training time under the CPU main memory storage scheme. In contrast, under the SSD storage scheme, data access becomes a significant bottleneck without tailored optimizations under the out-of-core training setup, comprising over 90% of the overall training time. Optimizing data access efficiency under the SSD storage scheme while considering the unique characteristics of T-GNN training is an interesting future direction for the DB community, as out-of-core setups can exhibit cost-efficient for training T-GNNs on large-scale dynamic graphs [6]. This approach may become increasingly practical as the scale of dynamic graphs in industry continues to grow.

Moreover, Table A3 provides a clear comparison of per-epoch training and data access times between ETC and TGL under all three storage schemes. Despite ETC's optimizations being primarily tailored for the CPU main memory storage scheme, our results indicate its effectiveness across all three storage schemes when compared to TGL.



(a) LastFM **(b) Wiki-Talk**

Figure A4: Per-epoch computation time comparison between the traditional and ETC's batching scheme under the same batch size.



(a) LastFM **(b) Wiki-Talk**

Figure A5: Epoch time breakdown under (1) GPU storage scheme, (2) CPU main memory storage scheme, and (3) SSD storage scheme by ETC and TGL.

3 Impact of Threshold Values.

We conduct case studies on the impact of threshold values on LastFM and Wiki-Talk using TGN model with a broad range of threshold values (500∼8000 with step size as 500). The results are shown in Figure A6.

Specifically, we oberseve the following findings: (1) Increasing the threshold value improves the model computational efficiency of T-GNNs. However, we notice an effect of diminishing marginal utility when the threshold becomes sufficiently large. Further increasing the threshold value beyond a certain point only results in minimal improvements in model computational efficiency. (2) Interestingly, lower threshold values do not necessarily guarantee better predictive performance. It is possible that the original dynamic graphs may contain some noisy interactions, which may result in an overfitted model [4]. Therefore, we recommend a reasonably large lower limit for the threshold value (e.g., over 1000) to mitigate the overfitting issue thus improving the generation ability of the underlying T-GNN models on the test set. On the other hand, with the threshold value increase to a certain point, the model performance can degrade drastically due to too much information loss. (3) Besides, we can see that there exists a broad range of threshold values that can present satisfactory runtime-accuracy tradeoff which allows practitioners to fine-tune the specific value based on their individual preference. For practitioners who prioritize higher model computation efficiency, choosing a relatively larger value within the range is advisable. Conversely, those aiming for better model performance should consider a relatively smaller threshold value within the range. (4) Moreover, we also notice the upper limit of the suitable threshold range can significantly vary on two datasets. In order to determine such an upper limit, it is important to consider the average update distance of the input dynamic graph (as illustrated in Table A4). The average update distance can reflect the update frequency for the node state vectors. For input dynamic graph with small average update distance such as LastFM, a small threshold value upper limit should be considered (e.g., around 3000). Otherwise,

Table A3: Comparison of per-epoch training time and data access time by ETC and TGL under three candidate storage schemes.

Dataset	Storage Type	Framework	Training	Data Access
LastFM	GPU memory	TGL	6.4	0.7
		ETC	6.0	0.3
	CPU main memory	TGL	16.7	6.1
		ETC	9.1	1.9
	SSD	TGL	252.4	234.6
		ETC	141.5	131.4
Wiki-Talk	GPU memory	TGL	35.7	1.6
		ETC	27.9	1.3
	CPU main memory	TGL	55.9	24.6
		ETC	35.2	7.3
	SSD	TGL	6959.3	6889.6
		ETC	5188.6	5120.6

Table A4: Summary of statistics of the dynamic graphs. $|V|$ and $|E|$ represent the number of nodes and edges. d_v and d_e denote the dimension of node features and edge features. α and β respectively represents the average degree and diameter. Θ denotes the average update distance of nodes. Specifically, given a node v with k times of update, the average number of interactions between each two of its k-time updates is defined the average update distance θ for node v. Then the average update distance for the whole graph is $\Theta = \sum_{v \in V} d_v/|V|$.

Dataset	$\|V\|$	$\|E\|$	d_v	d_e	α	β	Θ
LastFM	2K	1.3M	128	128	1306	1	2873
Wiki-Talk	1.1M	7.8M	172	172	14	11	458149
Stack-Overflow	2.6M	63.4M	172	172	49	13	1354646
GDELT	17K	191.3M	413	186	22934	7	4876113

a large threshold value upper limit is recommended (e.g., around 5000).

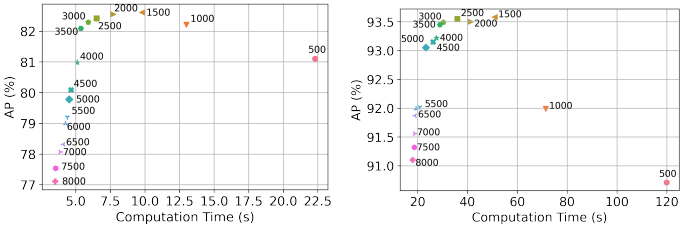
4 Compatibility of the Model-side Optimizations.

Regarding the observed accuracy gap between TGN [5] trained with ETC and TGN trained with Orca [2], this difference primarily stems from the subtle structural variance within the underlying models. More specifically, Orca [2] employs a more effective time encoder [1] within TGN's computation logic.

To investigate this further, we train the Orca-TGN model using ETC on LastFM, Wiki-Talk, and GDELT datasets, on which the original performance of the plain TGN model by ETC is not satisfactory. The same hyper-parameter settings in the main experiments are utilized. The results are presented in Table A5. Notably, when compared to using the plain TGN model, ETC achieves a substantial improvement in predictive performance, with enhancements of up to 10 points when using Orca's version of TGN. This experiment demonstrates the ability of ETC to incorporate the model-side optimizations for accuracy enhancement.

Table A5: Comparison of the average precision of the plain TGN and Orca-TGN by ETC on LastFM, Wiki-Talk, and Stack-Overflow.

Dataset	Model	AP(%)
LastFM	TGN	80.79
	Orca-TGN	**87.00 (+6.21)**
Wiki-Talk	TGN	93.41
	Orca-TGN	**96.67 (+3.26)**
Stack-Overflow	TGN	86.10
	Orca-TGN	**96.98 (+10.88)**



(a) LastFM **(b) Wiki-Talk**

Figure A6: The effect of different values of threshold ε. X-axis denotes the per-epoch model computation time (s) and Y-axis denotes the average precision (%).

References

[1] 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 *International Conference on Learning Representations*, 2023.

[2] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. Orca: Scalable temporal graph neural network training with theoretical guarantees. *Proceedings of the ACM on Management of Data*, 1(1):1–27, 2023.

[3] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. Zebra: When temporal graph neural networks meet temporal personalized pagerank. *Proceedings of the VLDB Endowment*, 16(6):1332–1345, 2023.

[4] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

[5] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML 2020 Workshop on Graph Representation Learning*, 2020.

[6] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. Mariusgnn: Resource-efficient out-of-core training of graph neural networks. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 144–161, 2023.

[7] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 international conference on management of data*, pages 2628–2638, 2021.

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

[9] 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. *Proc. VLDB Endow.*, 15(8), 2022.