TGM: A Modular Framework for Machine Learning on Temporal Graphs

Jacob Chmura [*1,2] **Shenyang Huang** [*1,2,3] **Ali Parviz** [1,4] **Farimah Poursafaei** [1,2] **Michael Bronstein** [3,5]
Guillaume Rabusseau [1,6,7] **Matthias Fey** [8] **Reihaneh Rabbany** [1,2,7]

Abstract

While deep learning on static graphs has been revolutionized by standardized libraries like PyTorch Geometric and DGL, machine learning on Temporal Graphs (TG), networks that evolve over time, lacks comparable software infrastructure. Existing TG libraries are limited in scope, focusing on a single method category or specific algorithms. We introduce Temporal Graph Modelling (TGM), a comprehensive framework for machine learning on temporal graphs to address this gap. Through a modular architecture, TGM is the first library to support both discrete and continuous-time TG methods and implements a wide range of TG methods. The TGM framework combines an intuitive front-end API with an optimized backend storage, enabling reproducible research and efficient experimentation at scale. Key features include graph-level optimizations for offline training and built-in performance profiling capabilities. Through extensive benchmarking on five real-world networks, TGM is up to 6 times faster than the widely used DyGLib library on TGN and TGAT models and up to 8 times faster than the UTG framework for converting edges into coarse-grained snapshots.

Code: tgm-team/tgm
Documentation: tgm.readthedocs.io

1. Introduction and Motivation

Temporal Graph Learning (TGL) has emerged as a crucial paradigm for modeling complex, time-evolving systems by capturing both spatial and temporal dependencies in dynamic networks (Cornell et al., 2025; Cao et al., 2020; Han et al., 2014). The field has seen explosive growth driven by high-impact applications across domains - from powering large-scale recommender systems (You et al., 2019) and social network analysis (Qiu et al., 2018) to enabling precise traffic forecasting (Yu et al., 2017; Li et al., 2018; Jiang et al., 2021) and pandemic response (Kapoor et al., 2020; Fritz et al., 2022; Lu et al., 2022). Unlike static graph approaches, TGL methods explicitly model temporal dynamics, making them uniquely suited for real-world systems where relationships evolve continuously. This capability has led to successful industrial deployments, such as LinkedIn's LiGNN system (Borisyuk et al., 2024) for user recommendations and mobility modeling that informed COVID-19 policy decisions (Chang et al., 2021).

Table 1. TGM uniquely combines support for both continuous-time (CTDG) and discrete-time (DTDG) dynamic graphs with scalability and a modular architecture.

Library	CTDG	DTDG	Scalable	Modular
TGM (ours)	✓	✓	✓	✓
DyGLib	✓	×	×	×
TGL	✓	×	✓	×
TGLite	✓	×	✓	✓
PyG Temporal	×	✓	✓	✓

However, the field's rapid methodological advances have outpaced its software infrastructure. While domains like computer vision and natural language processing benefit from mature, standardized libraries that accelerate both research and deployment, temporal graph learning remains fragmented. Researchers must often choose between implementing methods from scratch or adapting incomplete reference implementations, creating significant barriers to reproducibility and innovation. Recent systems research has produced valuable optimizations, from communication-efficient batch pipelining (Gao et al., 2024) and workload reduction techniques (Li et al., 2023) to redundancy-aware

[*]Equal contribution [1]Mila - Quebec AI Institute [2]School of Computer Science, McGill University [3]University of Oxford [4]New Jersey Institute of Technology [5]AITHYRA [6]DIRO, Université de Montréal [7]CIFAR AI Chair [8]Kumo.AI. Correspondence to: Jacob Chmura <jacob.chmura@mail.mcgill.ca>, Shenyang Huang <shenyang.huang@mail.mcgill.ca>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

Figure 1. High-level data flow of temporal graph training. The input edge stream is first split into batches. Next, temporal neighbourhoods are computed, and feature data for the relevant nodes is read from storage. Features are moved to the GPU and forwarded to a model for computation.

training (Wang & Mendis, 2023), but these advances remain isolated without a unified framework to integrate them.

We address this critical gap with TGM, a comprehensive framework designed specifically for temporal graph learning research and applications. Drawing inspiration from successful libraries like PyTorch Geometric, TGM separates high-level modelling abstractions from optimized low-level operations. This design enables researchers to focus on algorithm development while leveraging efficient implementations of core operations like temporal message passing, dynamic graph sampling, and neighbourhood computation. Designed for offline training settings, TGM simplifies preprocessing, enables flexible combinations of graph and model components, and lowers the entry barrier for new users. Its architecture promotes rapid prototyping, independent backend optimization, and fine-grained performance benchmarking, making it a robust platform for developing and evaluating dynamic graph algorithms.

Our key contributions include:

- A standardized machine learning library for temporal graphs that supports both continuous-time and discrete-time methods, as well as both dynamic link-prediction and node-property-prediction; the first of its kind.
- A carefully engineered API that decouples graph operations from model logic, enabling rapid experimentation while maintaining high performance.
- Comprehensive profiling tools that expose system bottlenecks and guide optimization efforts across the temporal graph learning pipeline.
- Extensive experiments across five datasets, demonstrating up to 6 times training efficiency gain over DyGLib for CTDG methods and up to 8 times speedup over UTG framework for the temporal coarsening procedure, often used in DTDG methods.

TGM is open-source and actively maintained, with comprehensive documentation and examples. TGM is available at `https://github.com/tgm-team/tgm` [1].

[1] `pip install tgm-lib`

2. Background and Related Work

This section provides a high-level overview of temporal graph learning, including the two formulations of dynamic graphs. For an overview of background concepts, we refer readers to Appendix A.

Definition 2.1 (Continuous Time Dynamic Graph (CTDG))**.** A CTDG $\mathcal{G}$ is represented as a stream of chronological edge events: $\mathcal{G} = \{(s_0, d_0, t_0), (s_1, d_1, t_1), ...\}$, where $0 \leq t_0 \leq t_1 \leq ...$ are timestamps and $s_i, d_i \in \mathcal{V}$ represent source and destination nodes for the temporal link $s_i \to d_i$ at time t_i.

Definition 2.2 (Discrete Time Dynamic Graph (DTDG))**.** A DTDG $\mathcal{G}$ is represented as a sequence of static graph snapshots sampled at regularly-spaced time intervals: $\mathcal{G} = \{G_0, G_1, ..., \}$, where $G_i = \{\mathcal{V}_i, \mathcal{E}_i\}$ is a static graph at snapshot i. As described in (Huang et al., 2024), a DTDG can be represented as a CTDG without loss of information by mapping each link in a given snapshot to a stream of events having the same time stamp.

Definition 2.3 (Dynamic Link Prediction)**.** Given a dynamic graph $\mathcal{G}$, and an edge event (s, d, t), learn time-aware representations $h_s^t, h_d^t \in \mathbb{R}^d$ using only historical interactions in $\mathcal{G}$ before time t to predict whether the link $s \to d$ will exist in $\mathcal{G}$ at time t.

Definition 2.4 (Dynamic Node Property Prediction)**.** Given a dynamic graph $\mathcal{G}$, and an node $s \in \mathcal{V}$ and timestamp t, learn time-aware representations $h_s^t \in \mathcal{R}^d$ using only historical interactions in $\mathcal{G}$ before time t to infer the state of s in $\mathcal{G}$ at time t.

TG Training Data Flow An overview of the data flow in temporal graph (TG) training is shown in Fig. 1. The input dynamic graph is stored on CPU and iterated in mini-batches. For each batch, temporal neighbourhoods are computed on-the-fly to enable time-aware message passing. Once neighbourhoods are identified, the corresponding node and edge features are materialized and transferred to the GPU. These tensors are then forwarded through the neural network for prediction.

Temporal Graph Libraries. Several libraries have been proposed to support temporal graph learning, including DyGLib (Yu et al., 2023b), TGL (Zhou et al., 2022),

DistTGL (Zhou et al., 2023), TGLite (Wang & Mendis, 2024), and TSL (Cini & Marisca, 2022). While DyGLib provides standard pipelines for continuous-time models, its limited scalability, lack of modularity, and weak support for discrete-time methods constrain its applicability (Gastinger et al., 2024). Table 1 summarizes the important aspects of these libraries. TGL and its distributed extension DistTGL have advanced large-scale sampling and multi-GPU execution, but lack an accessible, researcher-oriented interface and have seen limited recent updates. TGLite focuses on message-flow modeling for continuous-time TGNNs, and TSL addresses spatiotemporal methods on static graphs. In contrast, TGM is designed to support both continuous- and discrete-time paradigms with an emphasis on scalability, modularity, and ease of use—making it a more extensible and sustainable platform for TGL research and prototyping. A more detailed discussion of related work is provided in Appendix B.

3. TGM Features

We present an overview of TGM's key features, followed by a detailed discussion of its core abstraction and operators.

Dataset Integration: TGM streamlines experimentation by supporting standard dataset formats widely used in the community. It offers seamless integration with the Temporal Graph Benchmark (Huang et al., 2023; Gastinger et al., 2024), enabling direct loading of dynamic link prediction and node property prediction datasets by name. Custom datasets are also supported via *CSV* and *Pandas* (pandas development team, 2020). Details provided in Appendix D.

Unified Temporal Graphs: Inspired by prior work on bridging snapshot and event-based dynamic graphs (Huang et al., 2024), TGM supports both discrete and continuous-time paradigms within a unified framework. Central to this is the `TimeDelta` abstraction which manages timestamp indexing, allowing both communities to build on a shared foundation.

Model Support: TGM implements a range of temporal graph learning methods, from simple baselines like EdgeBank (Poursafaei et al., 2022a), which exploits historical edge patterns, to more expressive models like TGAT (da Xu et al., 2020) and TGN (Rossi et al., 2020), which incorporate temporal attention and memory. Ongoing development includes transformer-based models such as DyGFormer (Yu et al., 2023a), and random walk-based models like NAT (Luo & Li, 2022).

Test Suite, CI/CD & Performance Tools: TGM is developed with sound engineering practices, including unit and integration tests in a CI/CD pipeline to ensure reliability and reproducibility. We adopt semantic versioning to communicate changes clearly and catch regressions early. TGM includes a performance module that tracks GPU usage and throughput, with support for visualization tools like Snakeviz (Davis, 2012) and FlameProf (Bobrov, 2017) to guide informed optimization for evolving TGL workloads.

4. TGM Design

An overview of the TGM architecture is illustrated in Fig. 2, with a detailed description of each module provided in the following sections.

Example Workflow. The following is an example workflow of TGM on the `tgbl-wiki` dataset from TGB (Huang et al., 2023).

```python
from tgm import DGraph
from tgm.loader import DGDataLoader
from tgm.hooks import (
    NegativeEdgeSamplerHook,
    NeighborSamplerHook,
)

dg = DGraph("tgbl-wiki", split='train',
    device="cuda")
hooks = [
    NeighborSamplerHook(num_nbrs=20),
    NegativeEdgeSamplerHook(dg.num_nodes),
]
loader = DGDataLoader(dg, hook=hooks,
    batch_size=200)

for batch in tqdm(loader):
    pos_out, neg_out = model(batch)
    loss.backward()
```

Storage Backend The TGM storage backend provides a read-only, static interface for querying dynamic graph data-timestamps, edge pairs, features, and node IDs. Optimized for offline training, the backend uses caching to accelerate access and supports extensibility for particular hardware or access patterns (Zhang et al., 2021; Sha et al., 2017) in the future. Our default implementation uses a chronologically sorted COO format, enabling binary search over timestamps and linear scans within time windows. A temporal index cache further reduces query costs. We plan to add a Temporal Compressed Sparse Row (TCSR) backend (Zhou et al., 2022), which organizes time-aware neighbour lists in contiguous memory blocks. Though expensive to construct, TCSR aligns with TGM's immutable data assumption.

Graph Views and Lazy Slice Tracking. TGM exposes immutable graph views as the primary user interface. A `Graph View` holds a reference to the storage backend and contains a `TimeDelta` for temporal resolution, a `Slice Tracker` for subgraph operations, and a local cache for computed properties. The `Slice Tracker` records metadata about requested operations on the `Graph View`. Immutability guarantees that views are safe for concurrent

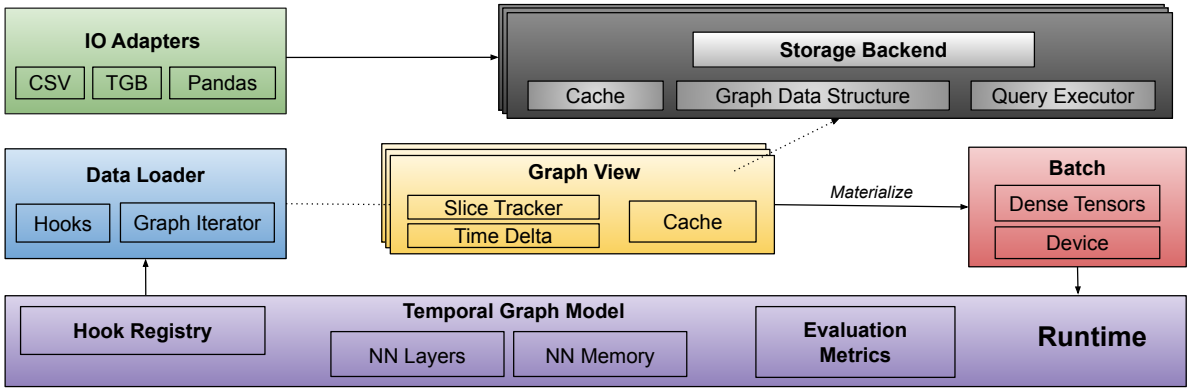

Figure 2. High-level TGM Architecture. Dotted lines denote *lazy* execution with *shallow* memory ownership. Solid arrows denote the passage of ownership and data materialization on-device. Users load a dynamic graph into a storage backend and register message-passing hooks. At runtime, the hooks execute and materialize the relevant sub-graph/feature vectors automatically when yielding a batch of data. Caching layers in both view objects and storage prevent duplicate computational work.

access. Read-only queries are delegated to the backend and automatically cached for efficiency. Subgraphs are lazily materialized at runtime, enabling optimized query execution while abstracting system-level optimizations behind a simple interface.

Temporal Index Management TGM uses a `TimeDelta` abstraction to define temporal granularity—whether timestamps represent ordered indices or real durations (e.g., seconds). This decouples time semantics from data layout and informs access patterns and iteration. The dataloader leverages `TimeDelta` to support batch iteration by event count (e.g., 200 events) or fixed time windows (e.g., 500 ms), enabling batching strategies that align with GPU efficiency or time-aware modelling, where event density varies—an approach that has demonstrated improved performance in recent dynamic link prediction tasks (Moritz Lampert, 2024).

Graph Hooks. TGM supports flexible prototyping via a modular hook mechanism. Hooks are composable transformations applied to sliced graph views during data iteration, enabling custom logic, such as memory updates or neighbourhood sampling, without modifying core code. Hooks we currently support include *Materialize*: converts a graph slice into dense edge index/feature tensors, *Negative Sampling*: generate negative edges for link prediction, and *Temporal Neighbourhood*: retrieves node interaction histories using online buffering or backend queries. We also implicitly include *Device* and *Deduplication* hooks. The former pins and transfers materialized tensors to the target device, enabling transparent device management for the user. The latter constructs an inverse index between global and batch-local node coordinates, reducing memory and computation overhead—particularly for large batches with many duplicate nodes. This inverse map allows users to seamlessly reference global or batch-local node embeddings without any manual management.

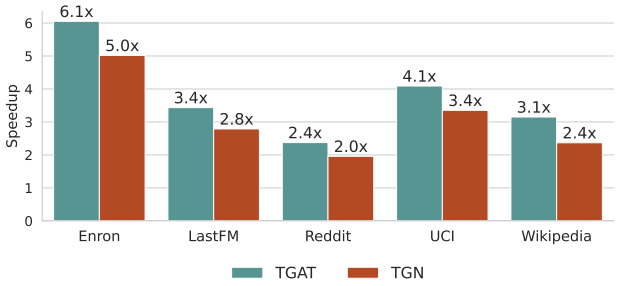

Figure 3. TGM speedup when compared to the commonly used DyGLib (Yu et al., 2023a) for CTDG method training.

Neural Layers & Runtime. TGM offers a PyTorch-Geometric-style frontend featuring modular components designed specifically for temporal graph learning, such as time encoders, memory modules, attention layers, and link decoders. Currently, it includes validated implementations of temporal self-attention and time encoding, with additional components planned for future releases. As our library grows and more models are supported, we anticipate expanding these components. Generic layers with broad applicability across multiple methods will be rigorously tested and integrated into the core library, while less established components will be added to an examples directory to encourage open research and keep the core library lean and maintainable.

5. Experiments

Experimental Setting. To empirically evaluate TGM, we benchmark several representative models from both CTDG and DTDG categories across five real-world datasets from (Poursafaei et al., 2022a). For CTDG methods, we

include TGAT (da Xu et al., 2020) and TGN (Rossi et al., 2020) while for DTDG methods, we include GCN (Kipf & Welling, 2017), and GCLSTM (Chen et al., 2018). The five benchmark datasets are Wikipedia, LastFM, Reddit, Enron, and UCI whose statistics and details are provided in Appendix D. Experiment details, model hyperparameters, and computational resources are highlighted in Appendix E. We focus on the dynamic link prediction task. Here, we highlight the efficiency of TGM when compared with alternative frameworks. Specifically, we compare against DygLib, which has emerged as the de facto standard among practitioners and researchers entering the field of temporal graph learning due to its broad model coverage and accessible design. Although more optimized frameworks exist, they are often narrowly tailored to specific model classes or graph settings and consequently exhibit lower adoption in practice. Given that TGM is explicitly designed to support a wide range of models and graph types while prioritizing usability and reproducibility, we consider DygLib to be the most appropriate baseline at this stage. As additional performance enhancements are incorporated, we intend to extend our benchmarking suite to include further specialized libraries. Test AUROC performance for all supported methods are reported in Appendix E.

CTDG Training Efficiency. To test the training efficiency of TGM, we compare with the popular DyGLib library (Yu et al., 2023a), designed only for CTDG methods. Fig. 3 shows the speedup of TGM when compared with DyGLib across five datasets for both TGN and TGAT methods. Notably, on the Enron dataset, TGM achieves 6.1x speedup with TGAT and 5.0x speedup with TGN. This shows that modular and optimized framework in TGM has significantly better efficiency than the DyGLib alternative.

Temporal Coarsening. As TGM decouples the time indices from the data layout, TGM allows for efficient temporal coarsening, the conversion from fine-grained edges into coarse snapshots. Fig. 4 shows the temporal coarsening time comparison between TGM and the alternative UTG framework (Huang et al., 2024). Fig. 4 shows that TGM is faster in temporal coarsening across all five datasets and 8 times faster on the UCI dataset. Temporal coarsening is an essential operation for DTDG methods.

6. Conclusion and Future Work

TGM is a modular, efficient, and extensible framework designed specifically for temporal graph learning, aiming to accelerate research and enable reproducible experimentation. By decoupling intuitive frontend interfaces from an optimized backend, it lowers the barrier to innovation while supporting SOTA methods. Looking ahead, our roadmap includes both algorithmic and system-level improvements, such as native memory modules and expanded model support. On the system side, we are developing a high-performance TCSR backend, native GPU storage, a C++ executor, and a runtime storage selector to adapt to varying workloads. TGM is deeply rooted in open-source principles.

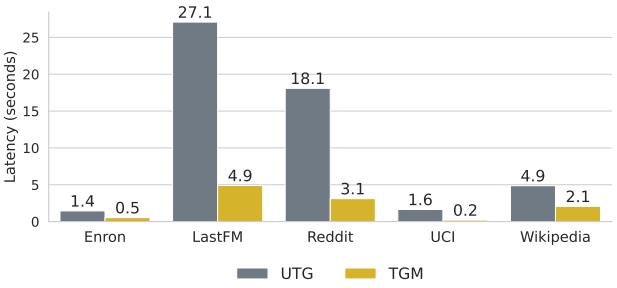

Figure 4. Comparison of temporal coarsening between the UTG framework (Huang et al., 2024) and TGM.

Impact Statement

This work introduces TGM, a unified and open-source framework that integrates continuous-time and discrete-time paradigms in temporal graph learning (TGL). By decoupling graph operations from model logic, TGM facilitates modular development, rapid prototyping, and system-level introspection through built-in profiling tools. These capabilities lower the barrier to entry for TGL research, particularly for those without access to large-scale infrastructure. TGM is optimized for single-GPU environments while remaining extensible to larger-scale deployments, promoting reproducibility and more equitable access to TGL experimentation across the research community.

We aim to foster a collaborative ecosystem where researchers and practitioners can contribute models, infrastructure, and ideas. By aligning with the open-source ML community, we envision TGM as a foundational tool for dynamic graph learning, bridging research and practice through shared, extensible infrastructure.

Acknowledgements

This research was supported by the Canadian Institute for Advanced Research (CIFAR AI chair program), the EPSRC Turing AI World-Leading Research Fellowship No. EP/X040062/1 and EPSRC AI Hub No. EP/Y028872/1. Shenyang Huang was supported by the 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. This research was also enabled in part by compute resources provided by Mila (mila.quebec).

References

Bobrov, A. Flameprof, 2017. URL https://github.com/baverman/flameprof.

Borisyuk, F., He, S., Ouyang, Y., Ramezani, M., Du, P., Hou, X., Jiang, C., Pasumarthy, N., Bannur, P., Tiwana, B., et al. Lignn: Graph neural networks at linkedin. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4793–4803, 2024.

Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., et al. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33:17766–17778, 2020.

Chang, S., Pierson, E., Koh, P. W., Gerardin, J., Redbird, B., Grusky, D., and Leskovec, J. Mobility network models of covid-19 explain inequities and inform reopening. *Nature*, 589(7840):82–87, 2021.

Chen, D., Li, Y., He, Y., Jin, X., and Tang, J. Gc-lstm: Graph convolution embedded lstm for dynamic link prediction. In *Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM)*, pp. 243–252. IEEE, 2018. doi: 10.1109/ICDM.2018.00038.

Cini, A. and Marisca, I. Torch Spatiotemporal, 3 2022. URL https://github.com/TorchSpatiotemporal/tsl.

Cornell, F., Smirnov, O., Gandler, G. Z., and Cao, L. On the power of heuristics in temporal graphs. *arXiv preprint arXiv:2502.04910*, 2025.

da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*, 2020.

Davis, M. Snakeviz, 2012. URL https://github.com/jiffyclub/snakeviz.

Fritz, C., Dorigatti, E., and Rügamer, D. Combining graph neural networks and spatio-temporal disease models to improve the prediction of weekly COVID-19 cases in Germany. *Scientific Reports*, 12(1):3930, 2022.

Gao, S., Li, Y., Shen, Y., Shao, Y., and Chen, L. Etc: Efficient training of temporal graph neural networks over large-scale dynamic graphs. In *Proceedings of the VLDB Endowment*, volume 17, pp. 1060–1072, 2024. doi: 10.14778/3641204.3641215. URL https://github.com/eddiegaoo/ETC. PVLDB Artifact Available.

Gastinger, J., Huang, S., Galkin, M., Loghmani, E., Parviz, A., Poursafaei, F., Danovitch, J., Rossi, E., Koutis, I., Stuckenschmidt, H., Rabbany, R., and Rabusseau, G. Tgb 2.0: A benchmark for learning on temporal knowledge graphs and heterogeneous graphs. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 140199–140229. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/fda026cf2423a01fcbcf1e1e43ee9a50-Paper-Datasets_and_Benchmarks_Track.pdf.

Giulio Rossetti, R. C. Community discovery in dynamic networks: a survey. *arXiv preprint arXiv:1707.03186*, 2019.

Han, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., and Chen, E. Chronos: a graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems*, pp. 1–14, 2014.

Huang, S., Poursafaei, F., Danovitch, J., Fey, M., Hu, W., Rossi, E., Leskovec, J., Bronstein, M., Rabusseau, G., and Rabbany, R. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems*, 36:2056–2073, 2023.

Huang, S., Poursafaei, F., Rabbany, R., Rabusseau, G., and Rossi, E. Utg: Towards a unified view of snapshot and event based models for temporal graphs. *arXiv preprint arXiv:2407.12269*, 2024.

Jiang, R., Yin, D., Wang, Z., Wang, Y., Deng, J., Liu, H., Cai, Z., Deng, J., Song, X., and Shibasaki, R. Dl-traff: Survey and benchmark of deep learning models for urban traffic prediction. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pp. 4515–4525, 2021.

Kapoor, A., Ben, X., Liu, L., Perozzi, B., Barnes, M., Blais, M., and O'Banion, S. Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv preprint arXiv:2007.03113*, 2020.

Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., and Brubaker, M. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1609.02907.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations*, 2018.

Li, Y., Shen, Y., Chen, L., and Yuan, M. Zebra: When temporal graph neural networks meet temporal personalized pagerank. In *Proceedings of the VLDB Endowment*, volume 16, pp. 1332–1345, 2023. doi: 10.14778/3583140.3583150. URL https://github.com/LuckyLYM/Zebra. PVLDB Artifact Available.

Longa, A., Lachi, V., Santin, G., Bianchini, M., Lepri, B., Lio, P., Scarselli, F., and Passerini, A. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *arXiv preprint arXiv:2302.01018*, 2023.

Lu, C., Han, T., and Ning, Y. Context-aware health event prediction via transition functions on dynamic disease graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4567–4574, 2022.

Luo, Y. and Li, P. Neighborhood-aware scalable temporal network representation learning. *Learning on Graphs Conference*, 2022.

Moritz Lampert, Christopher Blocker, I. S. From link prediction to forecasting: Addressing challenges in batch-based temporal graph learning. *arXiv preprint arXiv:2406.04897*, 2024.

pandas development team, T. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

Poursafaei, F., Huang, S., Pelrine, K., and Rabbany, R. Towards better evaluation for dynamic link prediction. *Advances in Neural Information Processing Systems*, 35: 32928–32941, 2022a.

Poursafaei, F., Huang, S., Pelrine, K., and Rabbany, R. Towards better evaluation for dynamic link prediction. *Advances in Neural Information Processing Systems*, 35: 32928–32941, 2022b.

Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., and Tang, J. DeepInf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2110–2119, 2018.

Rossi, E., Chambers, B., Ying, R., Bronstein, M., and Monti, F. Temporal graph networks for deep learning on dynamic graphs. In *Proceedings of the 2020 ICML Workshop on Graph Representation Learning and Beyond*, 2020. URL https://arxiv.org/abs/2006.10637.

Sha, M., Li, Y., He, B., and Tan, K.-L. Accelerating dynamic graph analytics on gpus. *Proceedings of the VLDB Endowment*, 11(1), 2017.

Shubham Gupta, S. B. A survey on temporal graph representation learning and generative modeling. *arXiv preprint arXiv:2208.12126*, 2022.

Wang, Y. and Mendis, C. TGOpt: Redundancy-aware optimizations for temporal graph attention networks. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, pp. 354–368. Association for Computing Machinery, 2023.

Wang, Y. and Mendis, C. Tglite: A lightweight programming framework for continuous-time temporal graph neural networks. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, pp. 1183–1199, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640414. URL https://doi.org/10.1145/3620665.3640414.

You, J., Wang, Y., Pal, A., Eksombatchai, P., Rosenburg, C., and Leskovec, J. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The world wide web conference*, pp. 2236–2246, 2019.

Yu, B., Yin, H., and Zhu, Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

Yu, L., Sun, L., Du, B., and Lv, W. Towards better dynamic graph learning: New architecture and unified library. *Advances in Neural Information Processing Systems*, 2023a.

Yu, L., Sun, L., Du, B., and Lv, W. Towards better dynamic graph learning: New architecture and unified library. *Advances in Neural Information Processing Systems*, 2023b.

Zhang, F., Zou, L., and Yu, Y. Lpma - an efficient data structure for dynamic graph on gpus. In Zhang, W., Zou, L., Maamar, Z., and Chen, L. (eds.), *Web Information Systems Engineering – WISE 2021*, pp. 469–484, Cham, 2021. Springer International Publishing. ISBN 978-3-030-90888-1.

Zhou, H., Zheng, D., Nisa, I., Ioannidis, V., Song, X., and Karypis, G. Tgl: a general framework for temporal gnn training on billion-scale graphs. *Proc. VLDB Endow.*, 15 (8):1572–1580, April 2022. ISSN 2150-8097. doi: 10.14778/3529337.3529342. URL https://doi.org/10.14778/3529337.3529342.

Zhou, H., Zheng, D., Song, X., Karypis, G., and Prasanna, V. Disttgl: Distributed memory-based temporal graph neural network training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701092. doi: 10.1145/3581784.3607056. URL https://doi.org/10.1145/3581784.3607056.

A. Background

This section provides a high-level overview of temporal graph learning, including the two formulations of dynamic graphs, everyday tasks, and associated model components. We follow the treatment in (Huang et al., 2024).

A.1. Temporal Graph Learning Methods

Given that our paper proposes a new framework, the focus is orthogonal to new learning methods and architectures. Therefore, we only give a brief overview of the literature. Interested readers are encouraged to find more information in surveys like (Longa et al., 2023; Giulio Rossetti, 2019; Shubham Gupta, 2022).

A.1.1. Temporal Message Passing

The temporal message passing framework is a neighbourhood aggregation scheme which recursively computes a latent representation by forwarding messages to temporal neighbours. Formally, if $\mathcal{N}^k(s)$ denotes the k-hop neighbourhood of node s in the dynamic graph $\mathcal{G}$, then the *temporal neighbourhood* $\mathcal{N}_t^k(s)$ is given by restricting neighbours to edge events chronologically before time t:

$$\mathcal{N}_t^k(s) = \{(s, d, t') \in \mathcal{N}^k(s) : t' \leq t\} \tag{1}$$

The combination of temporal and topological constraints makes efficient neighbourhood particularly challenging, requiring complex hierarchical data structures and cache-aware programming to sustain high-throughput on GPU stream multiprocessors (Zhang et al., 2021; Sha et al., 2017). We bypass the insertion and deletion complexity by assuming the entire graph structure is read-only. Temporal message proceeds by creating and passing messages between such sub-neighorhoods:

$$m_s(t) = msg(h_s(t), h_d(t), e_{s,d,t}) \tag{2}$$
$$\hat{m}_s(t) = agg(\{m_d(t) : d \in \mathcal{N}_t^k(s)\}) \tag{3}$$
$$h_s^t = upd(h_s^t, \hat{m}_s(t)) \tag{4}$$

In particular, messages are created by concatenating embeddings, aggregating embeddings across temporal neighbourhoods, then updating the new hidden representation. Such information flow occurs concurrently for each event in a batch of data.

A.1.2. Time-Encoding and Memory-Based Learning

Time-encoding based models use a shift-invariant model $\psi : T \to \mathbb{R}^{d_t}$ that maps a real-valued time stamp into a d_t-dimensional vector (e.g. TGAT (da Xu et al., 2020) use time-encoders like Time2Vec (Kazemi et al., 2019)). This encoding is then passed through modified self-attention blocks or feedforward layers. *Memory-based* models, such as TGN (Rossi et al., 2020), utilize a fixed-bandwidth memory module that compresses relevant information for each node and updates it over time. EdgeBank (Poursafaei et al., 2022a) is a non-parametric, memory-based method that memorizes and predicts new links at test time based on their occurrence in the training data.

A.1.3. Neighborhood Sampling

A key graph operation involves downsampling the temporal neighbourhood $\mathcal{N}_t^k(s)$ to a fixed size. This enables hardware optimization and reduces memory requirements. Existing libraries typically implement the following samplers:

- *Uniform Sampler*: Uniformly down-sample from all possible neighbours.
- *Weighted Sampler*: Assigns lower sampling probability to temporally distant neighbours.
- *Recency Sampler*: A FIFO-queue based data structure that stores each node's most recent k events.

TGM supports all sampler types, but the focus is given to *recency* and *uniform* samplers since they are the canonical methods in the literature. Due to the sampling operations, data access patterns are inhomogeneous, making coalesced memory access challenging. Therefore, most methods operate under the paradigm of *hybrid CPU-GPU* layouts (Gao et al., 2024) where features and graph indices are stored and sampled on CPU, then materialized and moved on-device. Figure 1 shows the high-level data flow during training.

B. Additional Related Works

DyGLib (Yu et al., 2023b): DyGLib offers standardized pipelines and evaluation scripts for several TGL methods, but its focus on continuous-time models limits support for discrete-time approaches and hampers scalability on large datasets (Gastinger et al., 2024). Its non-modular codebase also makes extensions difficult, contributing to low development activity. In contrast, OpenDG supports both temporal paradigms and is built for scalability, offering a more sustainable foundation for the TGL community.

TGL (Zhou et al., 2022): OpenDG builds on the scalability advances of TGL, which introduced an efficient temporal neighbourhood sampler capable of handling billion-scale graphs. While TGL has not been updated in relatively long period of time (over two years) and lacks a researcher-oriented interface, OpenDG addresses these gaps with a user-friendly API and a design centered on extensibility.

DistTGL (Zhou et al., 2023): DistTGL extends TGL to GPU clusters using batch pre-fetching and pipelining to reduce communication overhead, making it ideal for production deployments. In contrast, OpenDG targets research and prototyping on single-node GPU setups, prioritizing usability, extensibility, and rapid development of TGL methods.

TGLite (Wang & Mendis, 2024) TGLite is a lightweight framework for continuous-time TGNNs, offering abstractions and performance optimizations for message-flow modelling. While complementary, OpenDG extends support to discrete-time models and plans to integrate TGLite's redundancy-aware optimizations to enhance performance.

TSL (Cini & Marisca, 2022): TSL is a well-maintained library for spatiotemporal methods on static graphs. In contrast, OpenDG targets dynamic graphs with evolving topologies, requiring specialized data structures, pipelines, and models to address their unique challenges.

C. Additional Results

Table 2. Test set Mean Average Precision (mAP) scores for discrete-time and continuous-time models across five datasets. Reported numbers are means over 5 runs with 2σ error bars corresponding to a 96% confidence interval.

Category	Model	Enron	LastFM	Reddit	UCI	Wikipedia
CTDG	Edgebank	0.854 ± 0.002	0.832 ± 0.001	0.957 ± 0.000	0.755 ± 0.002	0.906 ± 0.000
	TGAT	0.729 ± 0.009	0.648 ± 0.019	0.952 ± 0.010	0.749 ± 0.090	0.855 ± 0.014
	TGN	0.732 ± 0.017	0.662 ± 0.041	0.936 ± 0.020	0.719 ± 0.082	0.864 ± 0.011
DTDG	GCN	0.627 ± 0.016	0.597 ± 0.007	0.987 ± 0.002	0.651 ± 0.015	0.847 ± 0.005
	GC-LSTM	0.642 ± 0.015	0.632 ± 0.009	0.657 ± 0.072	0.529 ± 0.037	0.713 ± 0.289

D. Dataset Details

In this work, we conduct experiments on Wikipedia (obtained from the TGB (Huang et al., 2023), where the dataset can be downloaded along with the package from TGB website), Reddit, LastFM, UCI, and Enron datasets (obtained from (Poursafaei et al., 2022b); these can be downloaded from `https://zenodo.org/records/7213796#.Y8QicOzMJB2`). These datasets span a variety of real-world domains, providing a broad testbed for evaluating temporal graph models. Detailed information about these datasets are as follows.

- **Wikipedia** is a bipartite interaction network that captures temporal editing activity on Wikipedia over one month. The nodes represent Wikipedia pages and their editors, and the edges indicate timestamped edits. Each edge is associated with a 172-dimensional LIWC feature vector derived from the edited text.
- **Reddit** models user-subreddit posting behavior over one month. Nodes are users and subreddits, and edges represent posting requests made by users to subreddits, each associated with a timestamp. Each edge is associated with a 172-dimensional LIWC feature vector based on post contents.
- **LastFM** is a bipartite user–item interaction graph where nodes represent users and songs. Edges indicate that a user listened to a particular song at a given time. The dataset includes 1000 users and the 1000 most-listened songs over a

Table 3. Dataset statistics.

Dataset	# Nodes	# Edges	# Unique Edges	# Unique Steps	Surprise	Duration
Wikipedia	9,227	157,474	18,257	152,757	0.108	1 month
Reddit	10,984	672,447	78,516	669,065	0.069	1 month
LastFM	1,980	1,293,103	154,993	1,283,614	0.35	1 month
UCI	1,899	26,628	20,296	58,911	0.535	196 days
Enron	184	10,472	3,125	22,632	0.253	3 years

Table 4. Hyperparameters used for each model

Parameter	**Edgebank**	**TGAT**	**TGN**	**GCN**	**GCLSTM**
Batch Size	200	200	200	Hourly	Hourly
Epochs	–	10	10	10	10
Learning Rate	–	1e-4	1e-4	1e-4	1e-4
Dropout	–	0.1	0.1	0.1	–
Number of Heads	–	2	2	–	–
Number of Neighbors	–	20	20	–	–
Embedding Dimension	–	100	100	128	128
Time Dimension	–	100	100	–	–
Sampling Strategy	–	Recency	Recency	–	–
Memory Mode	Unlimited	–	–	–	–
Number of Layers	–	–	–	2	2

one-month period. This dataset is not attributed.

- **UCI** is an anonymized online social network from the University of California, Irvine. Nodes represent students, and edges represent timestamped private messages exchanged within an online student community. The dataset does not contain node or edge attributes.
- **Enron** is a temporal communication network that is based on email correspondence over a period of three years. Nodes represent employees of the ENRON energy company, while edges correspond to timestamped emails. The dataset does not include node or edge features.

E. Compute Resources and Experiment Details

Compute: Each experiment was conducted on an Ubuntu 20.04 system with 8GB RAM, 2 isolated AMD EPYC 7502 CPU cores, and a single A100 GPU; we used SLURM to isolate environments and ensure no concurrent jobs were running.

Experiment Details: All models use default dataset splits from similar to the TGB benchmark (Huang et al., 2023; Gastinger et al., 2024). Hyperparameters for each model are shown in Tables 4. For event-based models (Edgebank, TGAT, TGN), training follows the native temporal structure. For discrete models (GCN, GCLSTM), we discretize data into hourly intervals and drop empty batches.

Test Performances. Table 5 shows the test AUROC scores for both discrete-time and continuous-time models across five real-world datasets. It shows that CTDG methods such as TGAT and TGN tend to achieve better performance than DTDG methods as they handles finer time granularities better.

Table 5. Test Set AUROC scores for discrete-time and continuous-time models across five real world datasets. Reported numbers are means over 5 runs with 2σ error bars corresponding to a 96% confidence interval.

Category	Model	Enron	LastFM	Reddit	UCI	Wikipedia
CTDG	Edgebank	0.887 ± 0.001	0.873 ± 0.000	0.957 ± 0.000	0.760 ± 0.001	0.907 ± 0.000
	TGAT	0.791 ± 0.006	0.673 ± 0.021	0.939 ± 0.013	0.696 ± 0.123	0.833 ± 0.013
	TGN	0.797 ± 0.012	0.678 ± 0.032	0.919 ± 0.024	0.656 ± 0.135	0.844 ± 0.007
DTDG	GCN	0.639 ± 0.023	0.627 ± 0.009	0.987 ± 0.004	0.636 ± 0.024	0.834 ± 0.011
	GC-LSTM	0.661 ± 0.020	0.689 ± 0.017	0.638 ± 0.041	0.526 ± 0.041	0.711 ± 0.312