Topological Preservation in Temporal Link Prediction

Marco Campos
Department of Mathematics *
University of Houston
Houston, TX 77204
mvcampo@sandia.gov

Casey Doyle
Sandia National Laboratories
Albuquerque, NM
cldoyle@sandia.gov

Daniel Krofcheck
Sandia National Laboratories
Albuquerque, NM
djkrofc@sandia.gov

Sarah Simpson
Sandia National Laboratories
Albuquerque, NM
sesimp@sandia.gov

Michael Xi
Sandia National Laboratories
Albuquerque, NM
michael.xi@rutgers.edu

William Ott
Department of Mathematics
University of Houston
Houston, TX 77204
william.ott.math@gmail.com

Henry Adams
University of Florida
Gainesville, FL 32611
henry.adams@ufl.edu

Abstract

Temporal link prediction seeks to model evolving networks to forecast future or missing interactions. Although many methods in this field achieve strong predictive performance, interpretability remains limited, especially in high-stakes domains. We address this by showing how topological data analysis can assess the faithfulness of learned representations to the underlying data, providing a pipeline for comparing temporal topological structure across model output. We further introduce a prototypical model that enables this analysis while maintaining predictive power. Taken together, these contributions lay the groundwork for models whose representations are more transparent to end users.

1 Introduction

Temporal link prediction models how relationships evolve in complex networks to forecast future connections and detect missing interactions, with applications spanning critical infrastructure and epidemiology. Recent advances in graph neural networks (GNNs)—notably graph convolutional networks (GCNs) and graph attention networks (GATs)—have improved inductive performance (see, e.g., Hamilton et al. [2018], Xu et al. [2020]). Interpretability, however, remains limited, obscuring how predictions arise from data and complicating causal or perturbation analyses in high-stakes settings.

In this paper, we show how the preservation of the latent topological and geometric structure of temporal graphs can be used to make link prediction models more transparent and trustworthy,

*Work done during their internship at Sandia National Laboratories.

Proceedings of the 1st Conference on Topology, Algebra, and Geometry in Data Science(TAG-DS 2025).

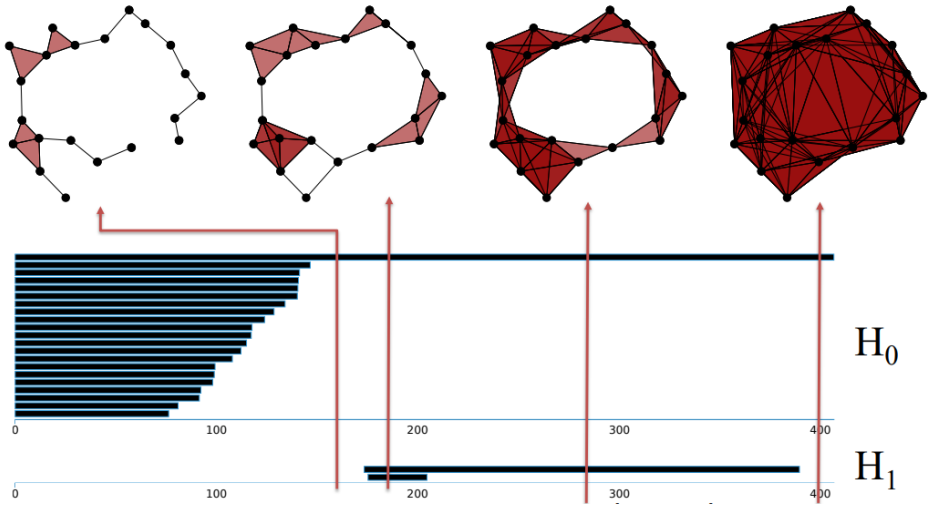

Figure 1: Top: Vietoris–Rips filtration on a point cloud, where increasing the scale parameter adds simplices until a complete graph is formed. Bottom: Corresponding barcodes for H_0 (connected components) and H_1 (cycles). Long bars indicate persistent features, here one connected component and one dominant cycle.

especially in datasets where persistent structures can be detected. We pair tools from topological data analysis such as persistent homology with temporal extensions (e.g., Carlsson and de Silva [2008], Carlsson et al. [2009], Myers et al. [2023]) to test if a model's embeddings capture the dynamic connectivity in real-world networks.

Building on these insights, we introduce a topology-preserving training paradigm that generates learnable disk graphs in latent space to preserve local connectivity and enhance global interpretability. We incorporate this strategy into a new architecture, the Temporal Disk Graph Network (TDGN), designed to easily assess topological fidelity throughout the training process. Our results demonstrate that preserving topological features in latent representations can simultaneously maintain predictive performance while clarifying the pathway from inputs to predictions.

2 Background

2.1 Temporal Link Prediction

Temporal link prediction (TLP) aims to forecast future or missing links in temporal graphs. Recent work has extended convolutional and attention mechanisms to temporal graph structures (Veličković et al. [2018]), as well as generalized them to the temporal domain (Sankar et al. [2019], Rossi et al. [2020], Xu et al. [2020]). TLP problems are typically categorized by how time evolves in the dataset. In the *discrete-time* setting, data is divided into equally spaced snapshots, while the *continuous-time* setting assumes an underlying dynamical system that generates links seamlessly over time. The discrete formulation can be viewed as a discretization of the continuous case, with temporal edges grouped into time steps. In practice, continuous-time snapshots may not be uniformly distributed, complicating batching.

Our methods using zigzag persistence (Section 2.2) are designed with an indexing set in mind. Therefore, we focus on the discrete version of the problem for our topological analysis while assuming we are batching over a continuously evolving process. Formally, we consider a series of observed snapshots of a graph over discrete timesteps $t \in \{1, \ldots, T\}$, denoted as $G = \{G^t\}_{t=1}^{T}$. Each snapshot $G^t = (V, E^t)$ is an unweighted undirected graph with node set V, link set E^t, and unweighted adjacency matrix A^t at time t. For weighted graphs, we process edge weights as features that the model uses to adjust embeddings.

We define G as a *temporal graph* and assume its node set V remains constant throughout all time steps. For simplicity, we assume no new nodes appear during the inference step. Most TLP algorithms aim

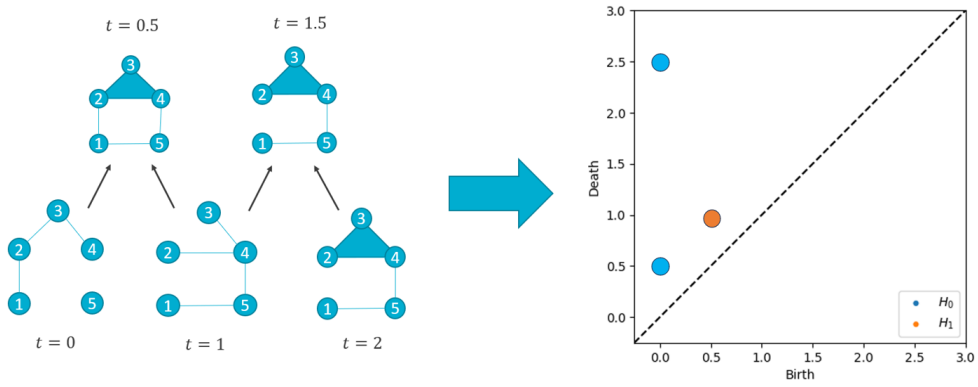

Figure 2: Left: A temporal graph evolving over three timesteps ($t = 0, 1, 2$) with intermediate Vietoris–Rips complexes ($t = 0.5, 1.5$) inserted to form a zigzag filtration. Right: The corresponding zigzag persistence diagram. It shows two connected components at $t = 0$, one of which merges with the main component at $t = 0.5$, leaving a single component persisting through all timesteps. A short-lived 1-cycle also appears at $t = 0.5$ and dies at $t = 1$, while smaller clique-based cycles are filled in and do not contribute.

to learn latent representations $e_v^t \in \mathbb{R}^d$ for each node $v \in V$ at time steps $t = 1, 2, \ldots, T$, using deep learning techniques followed by a final layer (e.g., a logistic regressor or Multi-Layer Perceptron) to evaluate the probability of an edge.

2.2 Topological Preliminaries

TDA uncovers the shape of data by detecting connected components, cycles, and voids in high-dimensional point clouds, offering local-to-global insights often missed by traditional methods. We provide a conceptual overview of methods that are relevant to our analysis, but for further details we direct interested readers to Carlsson and de Silva [2008], Carlsson et al. [2009], Dey and Wang [2022], Zomorodian and Carlsson [2005].

Data Structures Many TDA methods require data structures that extend beyond graphs to higher-dimensional analogues. Intuitively, these are obtained by "filling in" cliques of a graph with higher-dimensional shapes, forming what are known as simplicial complexes. A central example is the *Vietoris–Rips complex* $\mathrm{VR}_\epsilon(S)$, built from a point cloud $S \subset \mathbb{R}^n$ using a distance parameter ϵ. In graph settings, one can define distances using shortest paths and then construct a simplicial complex by filling cliques with simplices. Appendix B provides a review of simplicial complexes as well as additional technical details.

We use temporal clique complexes for the training data, and Vietoris-Rips complexes on learned embeddings to study topology at each timestep. When a consistent scale parameter exists, we directly compare temporal topology between embeddings and ground truth.

Persistent Homology Simplicial homology provides a method for detecting "holes" in simplicial complexes, capturing connected components, cycles, and higher-dimensional voids. For graphs, this unifies the detection of connected components and cycles into a single pipeline, avoiding the need for separate spectral methods. Persistent homology extends this framework to filtrations, sequences of nested simplicial complexes that track how features appear and disappear across scales, with outputs visualized as barcodes (Figure 1, bottom) or persistence diagrams (Figure 2, right) that provide concise topological summaries of static structures.

Zigzag Persistence For temporally evolving data, it becomes necessary to relax the usual filtration condition. Zigzag persistence achieves this by allowing inclusions in the filtration to proceed in either direction, extending persistent homology to capture temporal structures, with recent applications highlighting its growing importance (Adams and Carlsson [2014], Bernardoni et al. [2023], Myers et al. [2023], Shamsi et al. [2024], Topaz et al. [2015], Tymochko et al. [2020]). Figure 2, adapted from Myers et al. [2023], shows how intermediate Vietoris–Rips complexes connect successive

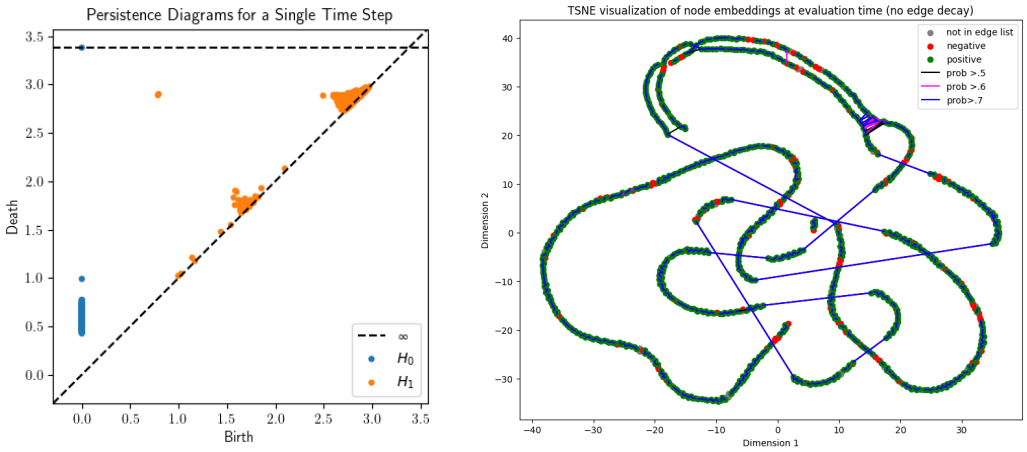

Figure 3: Left: Persistence diagram for a single timestep of the Vietoris-Rips complex generated from DySAT's temporal embeddings. Note, there are two overlapping points near $(0.7, 2.8)$, corresponding to the large cycles in the ground truth. Right: Visualization of node embeddings for the rotating sensors with edge probabilities between positive and negative samples generated by DySAT.

graphs, with the resulting zigzag persistence diagram tracking the birth and death of connected components and cycles across time (see Appendices B, D for further examples).

The persistence diagrams generated by zigzag persistence can be interpreted analogously to those from standard persistent homology, providing topological summaries of temporal features. For a simplicial complex indexed by time, one can insert intermediate complexes between adjacent timesteps to form a zigzag filtration (see Figure 2 for an example).

To construct temporal simplicial complexes from point cloud data, we use the Vietoris–Rips complex for its ability to detect higher-dimensional features. For example, a graph triangulating a torus contains only 1-cycles at the graph level, whereas the Vietoris–Rips recovers the global 2-dimensional toroidal void. By contrast, computing zigzag persistence directly on graphs often yields less intuitive birth–death times for clique-related cycles (see Myers et al. [2023], Fig. 3).

3 Experiments

Our experimental investigation was conducted in two phases. First, we utilize Dynamic Self-Attention Network (DySAT) Sankar et al. [2019] as an exemplar model to test for preserving temporal topology, attributed to its structure-preserving architecture. Based on these results, we then developed a model independent of discretization, retaining key structural elements of DySAT while simplifying topological analysis. This adaptation allowed us to replicate experiments without fixed graph snapshots or scale parameters when applying zigzag persistence.

3.1 Analysis of Learned Embeddings

We use the DySAT model as a baseline to test whether temporal topology can be identified in learned representations. DySAT adopts a discrete-time formulation well suited to our analysis, given its design emphasizes both temporal and structural dynamics. To further elucidate this point, an overview of the model's architecture is provided in Appendix A.

We developed a pipeline that follows Myers et al. [2023] and uses the Dionysus 2 package (Morozov [2019]) to generate zigzag persistence diagrams for the ground truth temporal graphs as well as any temporal graphs generated by our models during inference. We applied this to two deterministic temporal graphs: in the rotating network (example D.2), nodes placed on two circles rotate in opposite directions, yielding two persistent cycles; in the oscillating network (example D.3), the circles move horizontally and periodically overlap, producing changes in connected components and cycles.

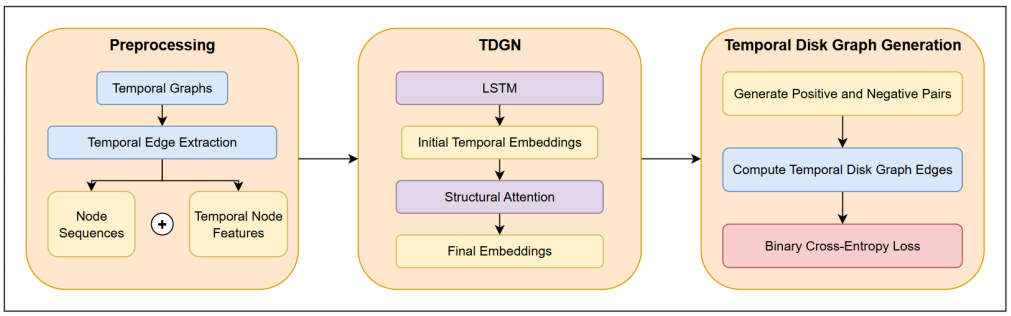

Figure 4: Pipeline describing the architecture of TDGN.

To align with benchmark practice, we formatted these datasets as temporal edge lists in the JODIE style (Kumar et al. [2019]). A custom wrapper for Dionysus converts these graphs into zigzag persistence diagrams by building intermediate transition graphs, forming Vietoris–Rips complexes, and computing persistence.

We then trained DySAT on the temporal graphs and analyzed the node embeddings with persistent homology. While DySAT preserves some structural features, it lacks a consistent scale parameter for constructing Vietoris–Rips complexes across all timesteps. This limitation motivated the development of our own model, which supports analysis with a fixed scaling parameter.

3.2 Topology-preserving models

To address the challenge of constructing temporal simplicial complexes from learned representations, we introduce the *Temporal Disk Graph Network* (TDGN), a discretization-independent model for link prediction on undirected graphs. TDGN fixes the scale parameter across timesteps, preserving temporal topology while producing interpretable embeddings. We use a loss function that anchors embeddings to both geometry and topology. This layer reconstructs the temporal graph as a high-dimensional geometric realization, enabling verification via zigzag persistence. Moreover, edge probabilities are computed directly from pairwise embedding distances, linking topology preservation with predictive performance.

The *learned disk graph* $\mathrm{VR}^1_{\beta^*}(S)$ for a vertex set S in metric space (X, d) is the collection of points $S \subset X$ together with edges $e^t_{u,v} = \{u_t, v_t\}$ for $u_t, v_t \in S$ such that $d(u_t, v_t) \leq 2 \cdot \beta^*$ for all pairs $u \neq v$ and learned parameter β^*. We adopt the notation $\mathrm{VR}^1_{\beta^*}(S)$, where the superscript implies restriction of the Vietoris-Rips complex to its 1-skeleton and the subscript corresponds to the disk radius parameter.

We construct disk graphs at each timestep for a temporal graph G with vertex set V using the learned node representations. By the Geometric Realization Theorem (Edelsbrunner and Harer [2010]), such embeddings exist in sufficiently high dimension. To score the likelihood of an edge between embeddings u_t and v_t, we define

$$p(u_t, v_t) = \frac{1}{1 + e^{\alpha(||u_t - v_t||_2 - \beta)}}. \tag{1}$$

The parameters $\alpha, \beta > 0$ are learnable, with β carrying geometric significance by dictating the radius of the temporal disk graph at inference. At inference, the final value β^* is fixed across all timesteps to simplify construction of the temporal disk graph. Training proceeds with a binary cross-entropy loss, encouraging embeddings to align so that the induced disk graph matches the temporal graph.

For TDGN, we use three modules: (1) a Temporal LSTM, (2) a Structural Attention layer, and (3) a Temporal Link Prediction head. The LSTM processes each node's temporal features $\{h^{t_i}_v\}^{t_M}_{t_i=t_0}$ as a multivariate time series, producing temporal embeddings $\{\ell^{t_i}_v\}^{t_M}_{t_i=t_0}$. The Attention layer then computes the final embeddings $\{e^i_v\}^{t_M}_{i=t_0}$ at each timestep via augmented graph attention (Appendix C, Eq. C.3). Finally, the Link Prediction head groups embeddings by timestep, builds the disk graphs, and computes the loss for backpropagation.

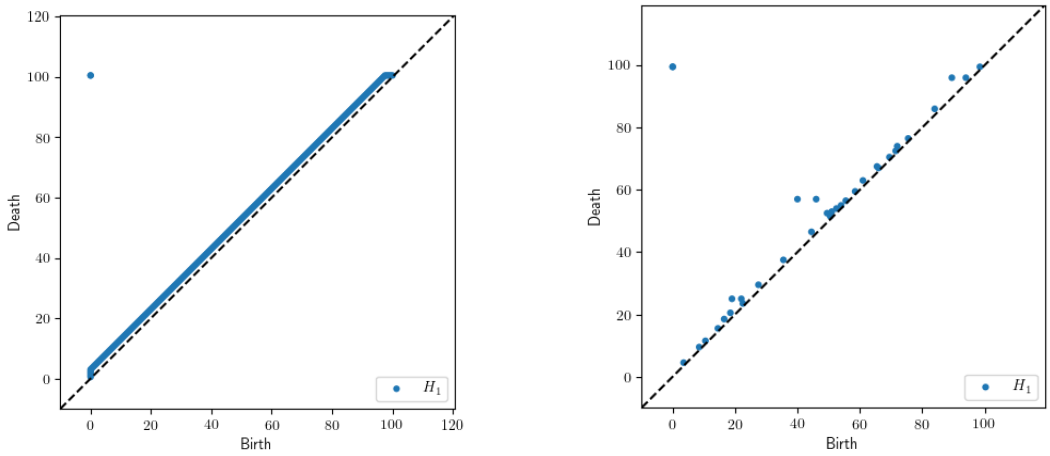

Figure 5: H_1 zigzag persistence diagrams for both the Vietoris-Rips complex constructed from the rotating sensor data (left) and the Vietoris-Rips complex built from the learned node embeddings from DySAT (right). Each diagram has two overlapping points near $(0, 100)$.

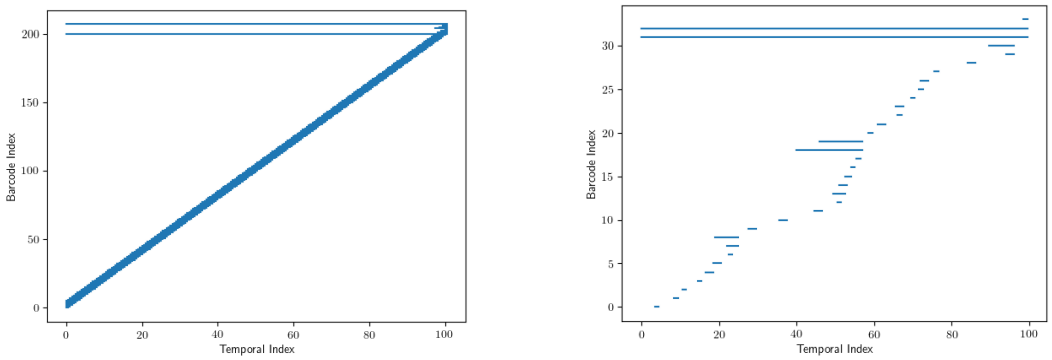

Figure 6: Corresponding barcodes for the complexes in Figure 10. These barcodes confirm the two persistent features, appearing in both complexes, one generated from the temporal graphs corresponding to the ground truth and the other from those generated by the temporal disk graph.

4 Results

4.1 Analysis of DySAT embeddings

Rotating Sensor Network Results First, we analyzed DySAT's temporal embeddings for the rotating sensor network described in Appendix D using persistent homology. The Vietoris-Rips complex revealed two persistent barcodes, shown in Figure 3 (left). The Vietoris-Rips complex revealed two persistent barcodes (Figure 3, left), corresponding to the two large cycles in the ground truth. At the final timestep, DySAT's edge probabilities also recover a high-dimensional structure resembling the two loops in the training data when thresholded at 0.7, as shown in Figure 3 (right).

To extend this analysis, we constructed Vietoris-Rips complexes for embeddings across all timesteps. We selected $\epsilon = 1.5$ as the distance parameter, since the persistence diagram indicated that connected components merged beyond this scale. Although more sophisticated techniques exist for parameter selection, this is beyond our current scope. The sensitivity of results to ϵ highlights a limitation in current temporal link prediction methods, which we revisit in Section 4.1.

Using zigzag persistence on the temporal complexes, we found that DySAT largely preserved the temporal topology of the training data. Figure 5 compares the one-dimensional persistence diagrams and barcodes of the training data and learned embeddings. Both reveal two persistent features corresponding to the large cycles, along with collections of near-diagonal points reflecting local belt-

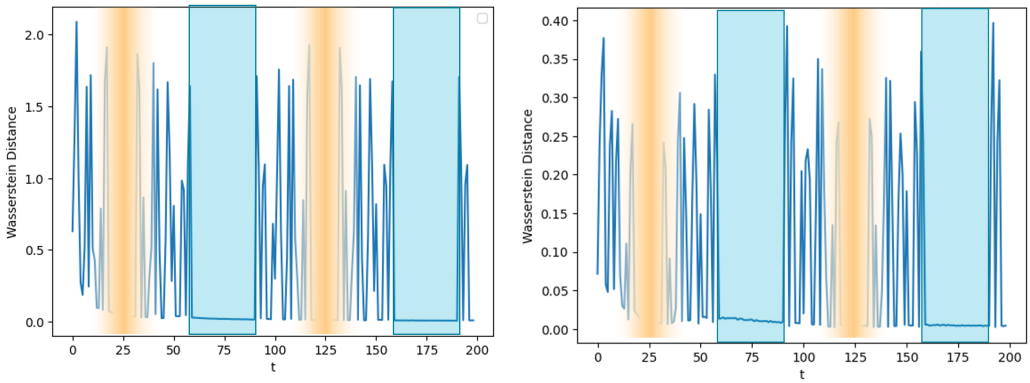

Figure 7: Pairwise 1-Wasserstein distances for persistence diagrams in dimensions 0 and 1. Orange shading corresponds to the size of the third cycle as it appears when the circles overlap. Blue regions indicate separations of the circles, creating an additional connected component.

like changes in the network. These results suggest that DySAT successfully captured the persistent elements of the temporal topology.

Oscillating Sensor Network Results In our second experiment, DySAT achieved strong performance (Table 1), but reconstructing dynamics from its embeddings proved difficult. Attempts to build disk graphs matching the ground-truth topology failed, since the required scale parameter was inconsistent across timesteps. Even with tuning, the resulting complexes emphasized only a few of the persistent features, yielding an averaged representation rather than true dynamics.

To investigate whether the training topology influences the learned representations, we computed persistence diagrams $\{B(t)\}_{t=t_0}^{t_n}$ from DySAT's embeddings and measured pairwise 1-Wasserstein distances $W_1(B(t_i), B(t_{i+1}))$ between consecutive timesteps (Figure 7). The pairwise distance decreases when the circles separate, reflecting topological stability, and gradually drops again as the third cycle grows—indicating that temporal topology continues to shape DySAT's learned embeddings.

4.2 Temporal Disk Graph Generation

Using the architecture from Section 3.2, our model achieves comparable performance to both DySAT and TGN for the artificial datasets, while also handling continuous-time datasets such as the Wikipedia dataset from TGBL Huang et al. [2023], Enron, and UCI without the need of a binning procedure. Table 1 reports results for our model against state-of-the-art continuous-time architectures, including DyRep Trivedi et al. [2018], DyGFormer Yu et al. [2023], TGAT Xu et al. [2020], and TGN Rossi et al. [2020], with many implementations drawn from DyGLib Yu et al. [2023]. TDGN performs competitively on the geometric datasets we developed, and we expect further optimizations in sequential data construction will enable scaling to larger datasets in the TGB benchmark Huang et al. [2023].

Beyond predictive performance, TDGN embeddings also recover topological features consistent with ground truth. In the Great Britain Transportation Network experiment Myers et al. [2023], Gallotti and Barthelemy [2015], where flights over a one-week period were modeled as temporal graphs, TDGN's persistence diagrams (Figure 13) show periodic clustering and de-clustering, reproducing the expected 0-dimensional features in zigzag persistence. The model also identifies a smaller persistent cycle, which we speculate reflects the dataset's weekly periodicity. Further details on temporal graph generation and persistence diagram comparisons are given in Appendix D.4.

5 Conclusion

Geometric and topological faithfulness in learned representations supports interpretability and enables the use of high-dimensional analysis tools, especially for relating temporal graphs to their underlying structure. Temporal link prediction arises in a wide range of applications such as modeling the

Table 1: Performance (AUC %) of TLP Models on training and inductive datasets. Cells marked † were not run for DySAT, as continuous-time datasets require the binning procedure of Sankar et al. [2019]. Top three results are highlighted: **red** for first, **blue** for second, and **bold** for third. TGNDisk denotes TGN with our geometric loss replacing the final MLP.

Model	Wikipedia		Enron		UCI		Rotating Cycles		Overlapping Cycles	
	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
TDGN (ours)	96.04	95.52	**99.03**	**99.09**	**94.21**	**94.34**	**98.82**	**98.43**	**98.59**	**97.69**
DySAT	†	†	†	†	†	†	**99.82**	**99.62**	92.44	88.76
DyRep	85.02	93.83	75.83	80.82	62.95	65.49	94.42	94.56	**94.70**	94.55
DyGFormer	**99.13**	**98.89**	**93.19**	**92.71**	**92.85**	**94.77**	**99.85**	**99.91**	**99.66**	**99.76**
TGAT	**97.47**	**96.55**	70.10	66.66	83.31	78.54	60.70	60.76	70.26	51.94
TGN	**98.36**	**97.76**	**89.41**	**88.15**	**90.37**	**92.99**	96.11	88.11	92.76	**94.84**
TGNDisk	96.83	95.56	87.55	82.39	79.56	79.92	95.99	80.34	92.82	83.58

spread of information or disease, anticipating interactions in biological or chemical networks, and forecasting connectivity in critical infrastructure systems. In such high-consequence settings, the ability to understand how and why a model produces its predictions is as important as achieving strong predictive accuracy.

In this work, we explored how topology-aware methods can contribute to this goal. First, we applied zigzag persistence to evaluate whether models retain temporal topological features, providing a direct probe of structural fidelity in evolving representations. Second, we proposed a link prediction paradigm that reconstructs node embeddings for temporal graphs which are spatially correlated to edge presence, offering a way to assess consistency between model outputs and the ground-truth dynamics. Third, we introduced the Temporal Disk Graph Network (TDGN), a prototypical architecture that preserves connectivity structure and facilitates the detection of zigzag persistence through learned embeddings.

Our results demonstrate that embeddings which preserve temporal topology are not only more interpretable but also more robust and theoretically grounded. Importantly, when embeddings exhibit spatial dependence they can be studied with the rich arsenal of statistical and geometric techniques developed for spatial and temporal data, further enhancing their value for downstream scientific analysis. These insights provide a first step toward building TLP models that balance predictive performance with topological interpretability and lay a foundation for future methods that combine topological fidelity with scalability to large temporal networks.

References

Henry Adams and Gunnar Carlsson. Evasion paths in mobile sensor networks. *The International Journal of Robotics Research*, 34(1):90–104, November 2014. ISSN 1741-3176. doi: 10.1177/0278364914548051. URL http://dx.doi.org/10.1177/0278364914548051.

William Bernardoni, Robert Cardona, Jacob Cleveland, Justin Curry, Robert Green, Brian Heller, Alan Hylton, Tung Lam, and Robert Kassouf-Short. Algebraic and geometric models for space networking, 2023. URL https://arxiv.org/abs/2304.01150.

Gunnar Carlsson and Vin de Silva. Zigzag persistence, 2008.

Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, SCG '09, page 247–256, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585017. doi: 10.1145/1542362.1542408. URL https://doi.org/10.1145/1542362.1542408.

Tamal K. Dey and Tao Hou. Fast computation of zigzag persistence, 2022. URL https://arxiv.org/abs/2204.11080.

Tamal Krishna Dey and Yusu Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2022.

H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. Applied Mathematics. American Mathematical Society, 2010. ISBN 9780821849255. URL https://books.google.com/books?id=MDXa6gFRZuIC.

Riccardo Gallotti and Marc Barthelemy. The multilayer temporal network of public transport in Great Britain. *Scientific Data*, 2(1):140056, January 2015. ISSN 2052-4463. doi: 10.1038/sdata.2014.56. URL https://doi.org/10.1038/sdata.2014.56.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. URL https://arxiv.org/abs/1706.02216.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997a. doi: 10.1162/neco.1997.9.8.1735.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, November 1997b. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

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, 2023. URL https://arxiv.org/abs/2307.01026.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2019.

Dmitriy Morozov. Dionysus 2, 2019. URL https://mrzv.org/software/dionysus2/.

Audun Myers, David Muñoz, Firas A Khasawneh, and Elizabeth Munch. Temporal network analysis using zigzag persistence. *EPJ Data Science*, 12(1), March 2023. ISSN 2193-1127. doi: 10.1140/epjds/s13688-023-00379-5. URL http://dx.doi.org/10.1140/epjds/s13688-023-00379-5.

Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *J. Am. Soc. Inf. Sci. Technol.*, 60(5): 911–932, May 2009. ISSN 1532-2882.

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

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks, 2019.

Kiarash Shamsi, Farimah Poursafaei, Shenyang Huang, Bao Tran Gia Ngo, Baris Coskunuzer, and Cuneyt Gurcan Akcora. Graphpulse: Topological representations for temporal graph property prediction. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=DZqic2sPTY.

Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. 2004. URL https://api.semanticscholar.org/CorpusID:59919272.

Chad M. Topaz, Lori Ziegelmeier, and Tom Halverson. Topological data analysis of biological aggregation models. *PLOS ONE*, 10(5):e0126383, May 2015. ISSN 1932-6203. doi: 10.1371/journal.pone.0126383. URL http://dx.doi.org/10.1371/journal.pone.0126383.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Representation learning over dynamic graphs, 2018. URL https://arxiv.org/abs/1803.04051.

Sarah Tymochko, Elizabeth Munch, and Firas A. Khasawneh. Using zigzag persistent homology to detect hopf bifurcations in dynamical systems. *Algorithms*, 13(11):278, October 2020. ISSN 1999-4893. doi: 10.3390/a13110278. URL http://dx.doi.org/10.3390/a13110278.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs, 2020. URL https://arxiv.org/abs/2002.07962.

Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library, 2023. URL https://arxiv.org/abs/2303.13047.

Afra Zomorodian and Gunnar Carlsson. Computing Persistent Homology. *Discrete & Computational Geometry*, 33(2):249–274, February 2005. ISSN 1432-0444. doi: 10.1007/s00454-004-1146-y. URL https://doi.org/10.1007/s00454-004-1146-y.

A DySAT Architecture

DySAT Sankar et al. [2019] has three main components: structural attention, temporal attention, and the graph context prediction. The model initially takes in the temporal graph G with T timesteps and generates latent representations of the nodes at every time $t \in \{1, \ldots, T\}$.

The structural attention layer consists of several self-attention layers based on the Graph Attention Network (GAT) formulation (Veličković et al. [2018]). This generates a sequence of node representations $\{h_v^1, \ldots, h_v^T\}$ for each vertex $v \in V$, where $h_v^t \in \mathbb{R}^f$ for embedding dimension f. The temporal attention block computes a set of position embeddings $\{p^1, \ldots, p^T\}$. The positions $p^t \in \mathbb{R}^f$ are then combined with the outputs from the structural attention block to get a new sequence $\{h_v^1 + p^1, \ldots, h_v^T + p^T\}$. The temporal self-attention layer computes a representation of node v at time t using the form of attention from Vaswani Vaswani et al. [2023]. This representation is finally passed through a feedforward layer to produce the final node representations $\{e_v^1, \ldots, e_v^T\}$.

To preserve structural and temporal information, DySAT employs an objective function that considers local neighborhoods around each node v at time t. This is achieved by collecting random walks $\mathcal{N}_{walk}^t(v)$ at each time t for each node v, along with n negative pairs $\mathcal{P}_n^t(v)$ which are unlinked nodes at time t according to a negative sampling ratio w_n.

DySAT uses a binary cross-entropy loss function to reward nodes close to each other in the same random walk and penalize negative pairs: In particular, for each node $v \in V$, we have:

$$L_v = \sum_{t=1}^{T} \sum_{u \in \mathcal{N}_{walk}^t(v)} -\log(\sigma(\langle e_u^t, e_v^t \rangle)) - w_n \cdot \sum_{u' \in \mathcal{P}_n^t(v)} \log(1 - \sigma(\langle e_{u'}^t, e_v^t \rangle)) \tag{2}$$

where σ is the sigmoid function. DySAT's primary objective is to produce node embeddings for separate inference. In Sankar et al. [2019], link prediction is performed by computing the Hadamard product $e_u^t \odot e_v^t$ of node pair representations (e_u^t, e_v^t) at time t, yielding a similarity score used to train a logistic regression classifier for each timestep.

B Mathematical Preliminaries

B.1 Simplicial Complexes from Point Clouds

Let V be a finite nonempty set. A *simplicial complex on* V is a collection K of nonempty subsets of V such that $v \in K$ for each $v \in V$, and if $\tau \in K$ and $\sigma \subset \tau$, then $\sigma \in K$. Each $\sigma \in K$ is called a *simplex*, with dimension $\dim \sigma = |\sigma| - 1$. The dimension of the complex is $\dim K = \max \dim \sigma \mid \sigma \in K$. In particular, simple graphs without self-loops or repeated edges are 1-dimensional simplicial complexes, where edges correspond to 1-simplices.

Consider a metric space (X, d) with metric d and a finite subset $S \subset X$ corresponding to some data within the metric space. In the case where we want to study the persistence of point clouds,

we let $S \subset \mathbb{R}^n$ be a collection of points in $\mathbb{R}^n$ with the Euclidean distance. We can now define the *Vietoris-Rips complex* $\mathrm{VR}_\epsilon(S)$ for the vertex set S with a parameter $\epsilon \geq 0$, where we add a simplex $\sigma = \{s_0, \ldots, s_k\} \subset S$ if and only if $d(s_i, s_j) \leq 2\epsilon$ for all i, j where $0 \leq i, j \leq k$. In other words, we add a simplex to a subset of points of S whenever all of the points are within a distance 2ϵ of each other. Moreover, we can use this to lift a graph G to a possibly higher-dimensional simplicial complex by allowing the distance $d(v_1, v_2)$ between vertices of the graph to be the length of the shortest path between them. In this case, Vietoris-Rips complex would correspond to the complex formed by all of the cliques in the graph G, commonly known as the *clique complex*.

B.2 Persistent Homology

Persistent homology computes features along a *filtration*, a finite, parametrized collection of nested simplicial complexes which are connected by inclusion maps.

$$K_0 \hookrightarrow K_1 \hookrightarrow K_2 \hookrightarrow \cdots \hookrightarrow K_n$$

In particular, this implies that our collection must be nested:

$$K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_n$$

For a given dimension $d \geq 0$ and simplicial complex K_i, we can generate the homology group $H_d(K_i)$ which is a vector space whose classes correspond to the d-dimensional holes in K_i. The inclusion maps then induce linear maps on these vector spaces which creates a *persistence module*, allowing us to study how the d-dimensional holes are changing across different K_i.

$$H_d(K_0) \to H_d(K_1) \to H_d(K_2) \to \cdots \to H_d(K_n)$$

This construction allows us to track births and deaths of topological features as we pass through the collection of simplicial complexes. Let γ be a d-dimensional feature born in the simplicial complex K_i which then dies in the later complex K_j. We then define the *persistence* of γ to be $\mathrm{pers}(\gamma) = j - i$. We can now visualize all of the features across the persistence module via two ways: the *persistence barcode* and *persistence diagram*. A persistence barcode $\mathrm{Bar}_d K$ consists of one bar for each d-dimensional feature γ, plotted as an interval along the x-axis starting at i and ending at j. Hence, the length of this bar is $\mathrm{pers}(\gamma)$. See Figure 1 for an example filtration of simplicial complexes with its corresponding barcode.

Persistence diagrams $\mathrm{Dgm}_d K$ are plots with the x and y axis corresponding to the birth and death times (i, j) of a given feature. These visualizations allow us to analyze how features "persist" across the collection of simplicial complexes as we increase the parameter, without having to visualize the space itself. However, note that a point (i, j) in the persistence diagram can correspond to multiple bars in its barcode representation. In this case, we say that the point (i, j) has some multiplicity m which corresponds to the number of features of dimension d with the same birth and death times. This can be effective when we want to discover how many connected components, cycles, voids, etc exist within a high-dimensional structure formed, for example, by the outputs of a neural network.

B.3 Zigzag Persistence

Persistent homology requires the collection of simplicial complexes to be nested in a filtration parametrized by an increasing scale parameter. Temporal graphs typically do not exhibit this behavior as we require the ability to add and remove edges. Therefore, we need a generalization. Fortunately, we can apply zigzag persistence which allows for the inclusions to switch directions. Note that the bidirectional arrows below can represent either a right inclusion map $\hookrightarrow$ or left inclusion map $\hookleftarrow$.

$$K_0 \leftrightarrow K_1 \leftrightarrow K_2 \leftrightarrow \cdots \leftrightarrow K_n$$

We will follow the construction outlined by Myers et al. [2023] for temporal graphs. Suppose $\{G_t\}_{t \in I}$ is a collection of graph snapshots for some temporal graph being sampled over a closed interval $I \subset \mathbb{R}$. We can define a collection of intermediate graphs $G_{t_i, t_{i+1}}$ for each t_i by computing the union of the graphs: $G_{t_i, t_{i+1}} = G_{t_i} \cup G_{t_{i+1}}$. For any graph G_{t_i} and $G_{t_i, t_{i+1}}$, with node set V, let d be

the shortest path distance on that graph. Namely, $d : V \times V \to \mathbb{R}$ where $d(u, v)$ is the number of edges needed to get from node u to node v with $d(u, v) = \infty$ if no path exists. This gives us a metric space construction on the vertex set of each graph and we can now apply the Vietoris-Rips complex with fixed scale parameter δ. The resulting sequence of simplicial complexes $K_{t_i} = \mathrm{VR}_\delta(G_{t_i})$ and $K_{t_i,t_{i+1}} = \mathrm{VR}_\delta(G_{t_i} \cup G_{t_{i+1}})$ inserted between the i and $i + 1$ terms creates a zigzag filtration of simplicial complexes as we have obvious inclusions towards the intermediate complexes:

$$K_{t_0} \hookrightarrow K_{t_1,t_2} \hookleftarrow K_{t_1} \hookrightarrow \cdots \hookleftarrow K_{t_{n-1}} \hookrightarrow K_{t_{n-1},t_n} \hookleftarrow K_{t_n}$$

One can interpret these new intermediate simplicial complexes as representing the "transition" steps between them. Alternatively, one could also perform this construction using the intersections of consecutive graphs which would then flip the direction of the inclusion maps at these intermediate steps.

We now have a *zigzag filtration* where each of our simplicial complexes is connected to its neighbors via an inclusion map. With the algebraic tools from Zomorodian and Carlsson [2005] and a clever application of Gabriel's Theorem from Representation Theory, we can use an analogous method to persistent homology called *zigzag persistence* which allows one to track topological features as we vary the parameter in I without requiring the simplicial complexes to be nested. We do so by applying the homology functor $H_d(\star)$ to the zigzag filtration. The inclusion maps will induce linear maps to form a zigzag persistence module which is a homological invariant of the zigzag filtration.

$$H_d(K_{t_0}) \to H_d(K_{t_0,t_1}) \leftarrow H_d(K_{t_1}) \to \cdots \leftarrow H_d(K_{t_{n-1}}) \to H_d(K_{t_{n-1},t_n}) \leftarrow H_d(K_{t_n})$$

The corresponding persistence barcode and persistence diagram can then be interpreted as the temporal topological features that persist in the homology groups $H_d(K_t)$ as the parameter t varies.

Note on zigzag persistence conventions: There are different conventions for the interval representations used to within the context of zigzag persitence. Consider the indexing set $S = \{1, \dots n\}$. When we work with persistence modules, we say a barcode corresponds to the half-open intervals $[b_j, d_j) \subset S$ for birth and death times b_j, d_j of the $j - th$ feature. In the original zigzag persistence paper Carlsson and de Silva [2008] by Carlsson and Zomoradian, a half open interval $[p, q)$ was instead represented as a closed interval $[p, q - 1]$ to respect the time-reversing symmetry. In this paper, we restrict attention to the forward direction of time, and therefore represent our intervals as half-open intervals $[b_j, d_j)$. As an example, the 1-dimensional feature in Figure 2 has the coordinate $(0.5, 1)$ in the persistence diagram and its barcode representation is the interval $[0.5, 1)$.

B.4 Topological Preservation

Suppose we have a learnable model f_θ such that, for each discrete time step $t \in \{1, \dots, T\}$, the function $f_\theta^t : V \to \mathbb{R}^d$ generates node representations $e_v^t \in \mathbb{R}^d$ for every node $v \in V$ of a temporal graph $G = \{G^t\}_{t=1}^T$.

Let $d_G : V \times V \to \mathbb{R}_{\geq 0}$ denote the shortest-path metric on each G^t, and let $d_E^t(u, v) := \|f_\theta^t(u) - f_\theta^t(v)\|_2$ denote the induced Euclidean distance between embedded nodes at time t.

Let $\mathrm{VR}_1(G)$ denote the temporal sequence of Vietoris–Rips complexes built from each G^t using the shortest-path metric with threshold 1. Similarly, let $\mathrm{VR}_\epsilon(f_\theta(V))$ denote the temporal sequence of Vietoris–Rips complexes constructed from the embeddings $f_\theta^t(V)$, using the Euclidean distance at a fixed scale $\epsilon > 0$.

Let $\mathrm{Dg}_{ZZ}^k(G)$ and $\mathrm{Dg}_{ZZ}^k(f_\theta(V))$ denote the k-dimensional *zigzag persistence diagrams* computed from these temporal filtrations, where the zigzag structure is induced via the construction described in Section B.

We define the *Wasserstein distance of order* $p \geq 1$ between two persistence diagrams E_1 and E_2 as:

$$W_p\left(\mathrm{Dg}_{ZZ}^k(E_1), \mathrm{Dg}_{ZZ}^k(E_2)\right) := \left(\inf_\gamma \sum_{(x,y)\in\gamma} \|x - y\|_\infty^p\right)^{1/p},$$

where γ ranges over all partial matchings between the two diagrams, allowing unmatched points to be paired with the diagonal.

We say the embedding function f_θ is δ-*Wasserstein stable in dimension* k if

$$W_p\left(\mathrm{Dg}^k_{ZZ}(G), \mathrm{Dg}^k_{ZZ}(f_\theta(V))\right) \leq \delta.$$

With these formalities addressed, we can then say a model is trying to *preserve the topology* of the underlying temporal graph in the sense that it is $\delta-$Wasserstein stable for a choice of δ. One of the main challenges that comes with this analysis is to identify an optimal choice of the scale parameter ϵ that minimizes this Wasserstein distance, i.e.,

$$\epsilon^* := \arg\min_{\epsilon>0} W_p\left(\mathrm{Dg}^k_{ZZ}(G), \mathrm{Dg}^k_{ZZ}(f_\theta(V)_\epsilon)\right),$$

where $f_\theta(V)_\epsilon$ denotes the embedding-induced complex constructed at scale ϵ. This allows us to select a scale at which the topological features in the embedded space are maximally aligned with those of the original temporal graph. Unfortunately, this scale parameter can be expensive to find in practice without performing experiments which require multiple computations of the Wasserstein distance for complexes at different scales. Therefore, we will also seek to build a model which freezes this scale and learns node embeddings whose pairwise distance in $\mathbb{R}^d$ can be used to form an inference graph which is used in the loss function to try and replicate the underlying temporal graph.

C Algorithms

We introduce the Temporal Disk Graph Network (TDGN), an architecture designed to effectively capture and integrate both temporal dynamics and structural information from graph-based data. The TDGN model is specifically tailored to process sequences of temporal features associated with nodes, including their respective timesteps, to generate meaningful temporal embeddings. This architecture is composed of two main modules: a Long Short-Term Memory (LSTM) Hochreiter and Schmidhuber [1997b] layer for capturing temporal behavior, and a structural attention layer that refines these temporal embeddings by incorporating structural context from the temporal graph. To train the TDGN model, we employ a modified Binary Cross-Entropy (BCE) loss with a scoring function which uses node embeddings that correspond to generating a unit disk graph.

We provide pseudocode for several algorithms we used for TDGN. We note that many of the details depend on our use of a similar version of the Jodie format, which is why algorithms such as Algorithm 1 depend on a table structure.

To clarify, we format the temporal edge set in the following way:

Table 2: Temporal Edge Set Format

Node 1	**Node 2**	**Timestep**	**Node 1 Features**	**Node 2 Features**	**Edge Features**
u	v	t	$\phi_u \in \mathbb{R}^{n_\ell}$	$\phi_v \in \mathbb{R}^{n_\ell}$	$\psi_{uv} \in \mathbb{R}^{e_\ell}$

We always specify the length of the node and edge features. For our experiments we assumed each of these had length 1, but the format allows for more general cases in more complicated applications.

C.1 Preprocessing Steps

1. **Data Splitting:** The temporal edge set is divided into training, validation, and test subsets. If using a TGBL dataset, we adopt their official splits; otherwise, we apply a $70\%/15\%/15\%$ chronological partition.
2. **Negative Sampling:** For each split, we generate negative edges that do not exist in the positive set using Algorithm 1. This ensures a balanced dataset of positive and negative examples.
3. **Dataset Construction:** Positive and negative samples are concatenated for each split, and binary labels are created (1 for positive edges, 0 for negative edges).
4. **Feature Dimensions:** Input size, edge feature size, hidden size, embedding size, and attention size are initialized according to model hyperparameters and the feature lengths.
5. **Sequence Preprocessing:** Each dataset split is passed through Algorithm 2, which organizes interactions into node-wise chronological sequences and applies padding to create uniform batch tensors for model input.

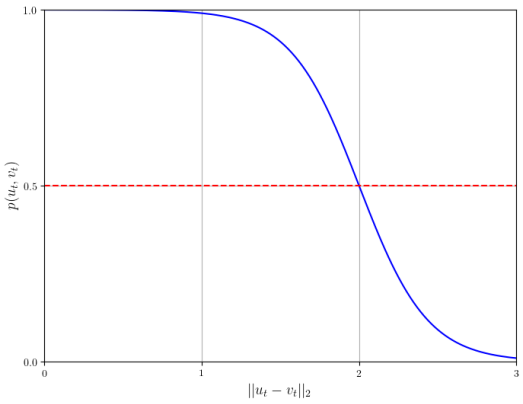

Figure 8: Visualization of the scoring function where in the x-axis, we plot the norm between two temporal embeddings and the y-axis gives a score which allows us to deduce when an edge in the unit disk graph should be present.

C.2 Node Embeddings Computation

C.2.1 LSTM

The LSTM within the TDGN architecture allows us to capture the temporal dynamics of node features across timesteps. Following the LSTM's processing, the output goes through a fully connected (linear) layer, which maps the LSTM-generated hidden states to an embedding space of a predetermined size.

It is worth noting that we previously experimented with an attention mechanism using position embeddings but we found the LSTM to yield better results although the authors plan to continue experimenting with the former due to their presence in state-of-the-art architectures and theoretical foundations.

C.2.2 Structural Attention

The structural attention layer refines the initial temporal embeddings through the incorporation of structural information from the graph. This is done through a self-attention mechanism Vaswani et al. [2023], designed to weigh the significance of each node's embedding in the context of its structural features and the overall graph topology. The model begins by concatenating the temporal embeddings, generated by the temporal attention module, with edge features specified by the given dataset. This combined input is then processed through a standard attention mechanism which a chosen embedding dimension as the output to generate the temporal embeddings for the construction of the unit disk graph. We note that we did not follow the formulation for GAT Veličković et al. [2018] as we are processing a data structure which is a collection of node features and not objects corresponding to the temporal graphs as a whole during a given snapshot which means we cannot utilize tools like the normalized graph Laplacian in the traditional way. Our approach offers a significant advantage by adopting a continuous-time methodology, which closely mirrors the manner in which real-life data is collected. Unlike traditional methods that require binning due to the presentation of data in time-indexed graphs, datasets like TGB are provided as edge lists with timestamps. This allows us to process the data in its natural, continuous form without the need for artificial segmentation.

C.3 Loss Function

Below is the binary cross-entropy loss function with the scoring function from Section 3.2 being used as inputs for a pair of nodes which allows for the construction of the unit disk graph.

$$L_v = \sum_{t=1}^{T} \sum_{u \in \mathcal{P}^t(v)} -\log\left(p(u,v)\right) - w_n \cdot \sum_{u' \in \mathcal{N}_n^t(v)} \log\left(1 - p(u',v)\right) \tag{3}$$

In this formulation, $\mathcal{P}^t(v)$ is the set of vertices that share an edge with v at time t, $\mathcal{N}_n^t$ is a collection of $n = |\mathcal{P}^t|$ negatively sampled nodes, and w_n is a weight assigned to the negative samples. A

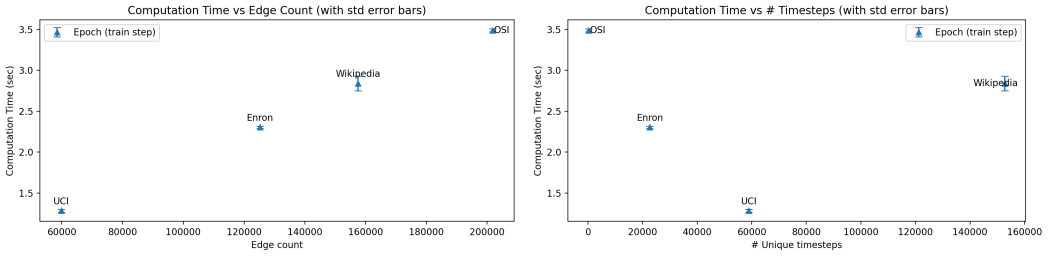

Figure 9: Computation Time for one training epoch vs. (left) edge count and (right) number of timesteps on four datasets (Wikipedia, OSI, Enron, UCI). Points show mean $\pm$ standard deviation times of a single epoch over 10 trials. For all datasets, TDGN was initialized with the same hidden dimension $= 100$ and embedding dimension $= 100$ (other hyperparameters held fixed). OSI is an abbreviation for Overlapping Sensor Network.

visualization of $p(u, v)$ in Figure 8 shows that at a distance of 2, we get a score of 0.5 which we consider the cut-off for when two nodes should share an edge due to the definition of the disk graph.

To compute the loss, we take the embeddings generated by our model which are ordered as node sequences in the form $\left[\{e_v^i\}_{i=t_0}^{t_{M_v}}\right]_{v=1}^{N}$ for N nodes with node id v over active time steps M_v. We then recompute the pairings for the positive and negatively sampled edges which were saved using our preprocessing Algrithm 3 and compute the loss according to Equation C.3. We then use Pytorch's ADAM optimizer to find the optimal parameters for the LSTM, Structural Attention, and disk computation.

C.4 Model Parameters

We report the hyperparameters used in our TDGN experiments in Table 3.

Table 3: Model Hyperparameters

MODULE	VALUE
LSTM EMBEDDING DIMENSION	100
LINEAR HIDDEN LAYER DIMENSION	100
STRUCTURAL ATTENTION DIMENSION	100

C.5 Computational Time

We analyze the computational complexity of our approach by examining both the zigzag persistence computations and the preprocessing of data for TDGN. TDGN is constructed using a standard LSTM Hochreiter and Schmidhuber [1997a] and an attention block Vaswani et al. [2023], so the complexity of its components follows from well-studied analyses of these architectures. Figure 9 gives an idea of how much time a single epoch takes to compute for datasets of various sizes. Although forward passes are fast, our current data-restructuring for edge-probability estimation is a major bottleneck. Future work will focus on accelerating the loss computation to enable experiments on larger TGB benchmark datasets. All experiments were conducted on a remote Linux server equipped with a single NVIDIA H100 GPU (80 GB memory) running CUDA 12.8.

Zigzag Persistence We computed zigzag persistence using the `Dionysus 2` library, which implements the original algorithm of Carlsson et al. with complexity $O(mn^2)$, where m is the zigzag length and n is the number of simplices. This is representative of most approaches in the literature, which typically exhibit cubic complexity due to repeated matrix reductions. More recent work has sought to improve these bounds, including near-quadratic methods and algorithms specialized for graph filtrations that achieve near-linear time. In particular, `FastZigzag` Dey and Hou [2022]

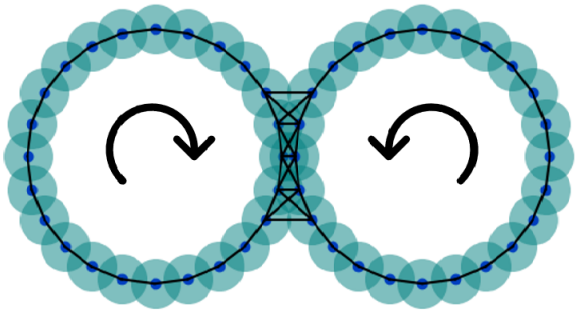

Figure 10: Visualization of a smaller version of the rotating sensor network data which generates the graph that DySAT is trying to learn. The evolving belt in the center creates enough dynamic changes to makes temporal embeddings nontrivial if we do not use methods that are structure preserving.

achieves $O(m^\omega)$ complexity where ω is the matrix multiplication exponent by exploiting dynamic data structures, providing a promising avenue for scaling zigzag persistence to larger datasets. We have not yet integrated `FastZigzag` since we already maintain a working wrapper API for `Dionysus 2`, but we anticipate transitioning to this package in future work to take advantage of these efficiency gains.

D Datasets

Table 4: Statistics for the datasets used in our experiments.

NAME	#NODES	#EDGES	#STEPS
WIKIPEDIA	9,227	157,474	152,757
ENRON	184	125,235	22,632
UCI	1899	59,835	58,911
ROTATING SENSORS	1,000	1,031,000	100
OSCILLATING SENSORS	1,000	201,808	200
GBTN-FLIGHTS	48	11496	1008

We provide a detailed overview of a collection of datasets integral to our study. Most of these datasets are loaded as edge lists in CSV files, adhering to a format analogous to the JODIE format where in addition we allow for the inclusion of node and edge features separately. Our model can also utilize datasets from the TGB library although wikipedia has been the only one we have experimented with due to its size.

D.1 Datasets from TGB and DyGLib

We utilized the wikipedia dataset from the TGB librairy Huang et al. [2023] as well as Enron and UCI which are thoroughly explored in Yu et al. [2023].

- Wikipedia (TGB)Huang et al. [2023].
  This dataset contains a bipartite interaction network from Wikipedia, tracking co-editing activities between editors and wiki pages over a month. Nodes represent editors and pages, and edges denote edits by a user on a page which are timestamped and include text from the edits as features.
- Enron Shetty and Adibi [2004]
  The Enron dataset contains around 50K emails exchanged among employees of the ENRON energy company over a three-year period. It does not contain attributes and further statistical details can be found in Shetty and Adibi [2004].

- UCI Panzarasa et al. [2009]
  This is an online communication network consisting of university students sending messages on an online forum.

D.2 Rotating Sensor Network

Consider a collection of n sensors with detection radius ϵ arranged into two circles separated by a distance of $\frac{\epsilon}{2}$ at their closest points. The sensors along the circles are separated by a distance of ϵ (see Figure 10). We generate edges based on whether two sensors have overlapping sensing regions. We allow these sensors to rotate so that while their connectivity data along the circles stays the same, there will be a dynamically evolving belt of sensors connecting between the two circles. This creates deterministic dynamics in two dimensions governed by a single parameter. The dynamics should be easy to learn given enough training samples to capture where and when edges appear between the two circles.

Figure 5 (Left) gives us the 1-dimensional zigzag persistence of the sensor network. The persistence diagram on its own is a bit deceiving as it appears to have only one persistent cycle but in reality, there are two overlapping points which can be seen by the two persistent bars below. The collection of points just off the diagonal corresponds to the rapid birth and deaths of the cycles in the belt of the rotating sensor network.

D.3 Oscillating Sensor Network

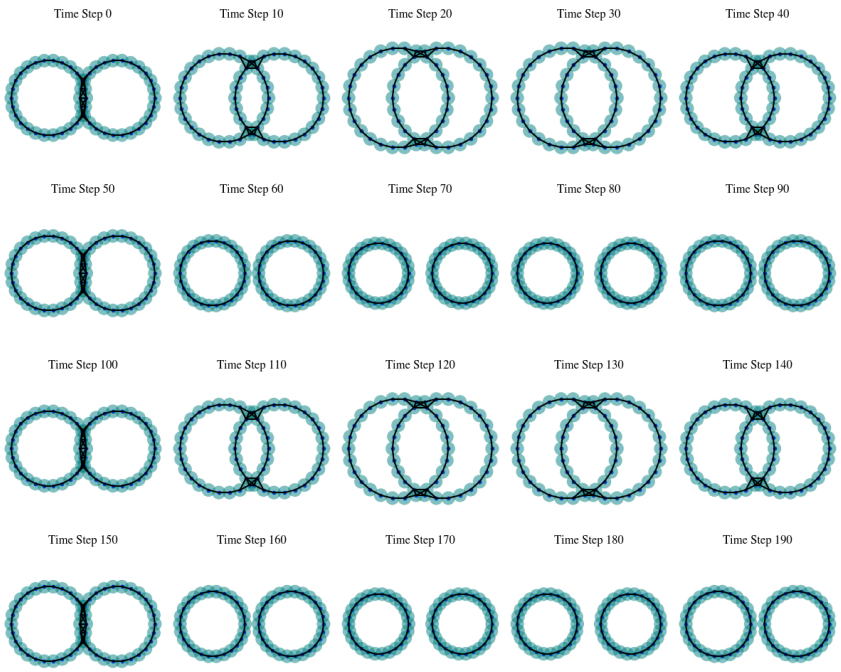

Figure 11: Visualization of a smaller version of the sensor network where the sensors belong to two groups of circles and shift periodically side-to-side. Here we have the number of connected components changing periodically during the splits and we have a third persistent cycle appearing periodically in the overlapping sections. This figure only shows 104 nodes per timestep but in our actual simulation, we used 1000.

The initial experiment of the rotating sensors is quite rigid and the main topological features are persistent across the entire simulation. Therefore, it is worth exploring what happens when we have persistent topological features that vary periodically.

We construct a simulation of 1000 sensors that once again splits sensors into two groups arranged into two circles but this time we have the circles move horizontally in an oscillating motion. The

oscillating motion has a a period of 100 timesteps, and we run the simulation for 200 timesteps to both create and destroy connections. The network periodically becomes disconnected when the two circles move far enough apart: it disconnects at around $t = 55$, reconnects at about $t = 95$, then disconnects again near $t = 155$ before connecting again at $t = 195$ near the end of the simulation. Additionally, when the circles overlap enough, there will be a third persistent cycle that will appear in the intersection. This cycle initially starts as a small cycle born from the belt between the two intersecting circles and grows gradually before shrinking back again and ultimately dying when the belt becomes filled in.

Figure 11 shows the dynamics of every 10^{th} snapshot for 200 timesteps for a smaller collection of sensors to help with visualization of the dynamics while Figure 12 showcases the zigzag persistence of the training data. Comparing the snapshots of the dynamics with the persistence diagrams allows one to see when the network disconnects and reconnects as well as when the third long cycle appears and disappears, without the need to form a complete visualization.

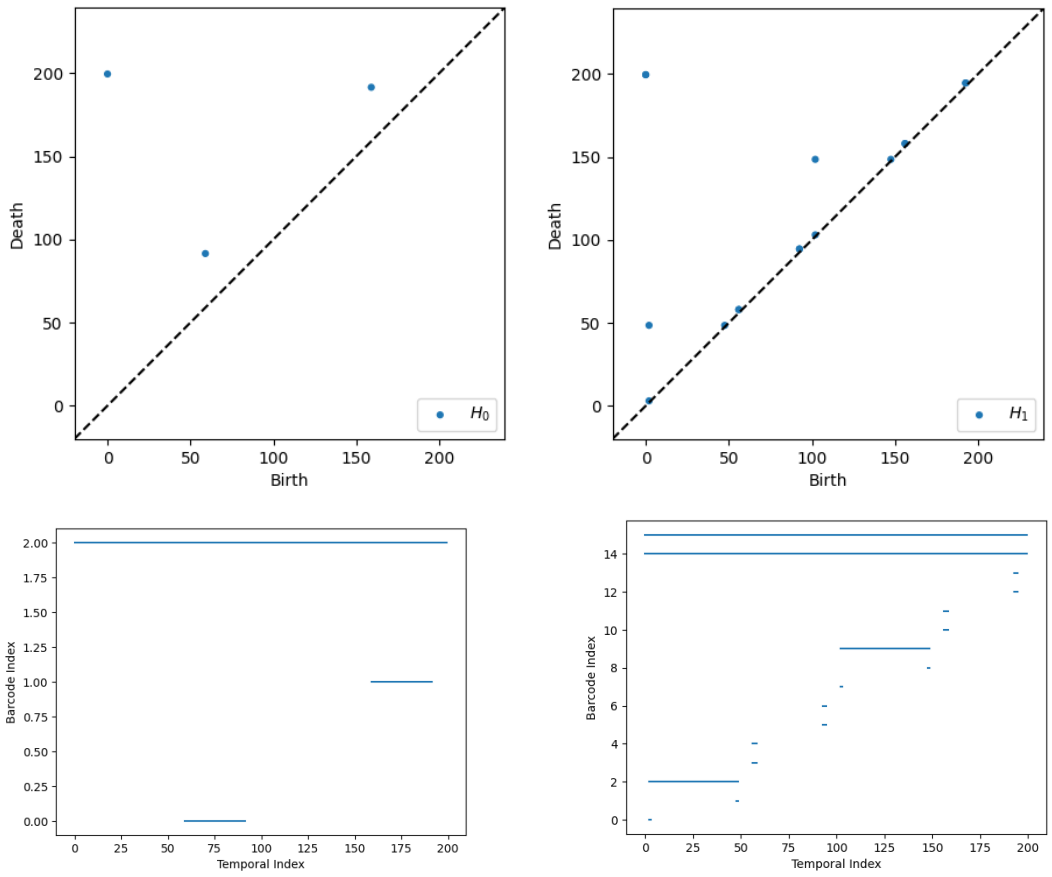

Figure 12: Left column: H_0 zigzag persistence diagram of the sensor network in Figure 11 with the corresponding barcode. Right column: The H_1 diagram and its barcode.

D.4 Great Britain Transportation Network

For this dataset, we generate temporal networks using the Great Britain temporal transportation dataset Gallotti and Barthelemy [2015] using a similar data curation as Myers et al. [2023] as they showed that this dataset exhibited noticeable periodic behavior when computing its zigzag persistence. To train our model, we focused on the air datasets which contained flight information for one week as its size was reasonable enough to test our link prediction capabilities. The dataset provides the destinations as nodes and a list of events between two stations. It is worth noting that the data comes in a multilayered format where one station could include events between multiple forms of transportation so we restricted ourselves to only data involving events which explicitly listed air

travel. Our construction of the temporal graph varies slightly as we pre-processed the data ourselves and settled on a set of parameters based on the statistics of the data set. In particular, we set a sliding window with a width of 60 minutes and a overlap of 50% between adjacent windows.

Figure 13 (left) showcases the zigzag persistence of the dataset which shows periodic behavior due most likely due to the same cycles being formed as the schedule of flights repeats daily. Figure 13 shows the zigzag persistence of the node emebddings generated for our model for the training data which consists of the first 5 days of the dataset. The learned embeddings not only exhibit periodic behavior, but also show a small persistent cycle being born on the top left corner of the diagram which while not present in the training data, could imply that the model is trying to enforce stronger periodic behaviors.

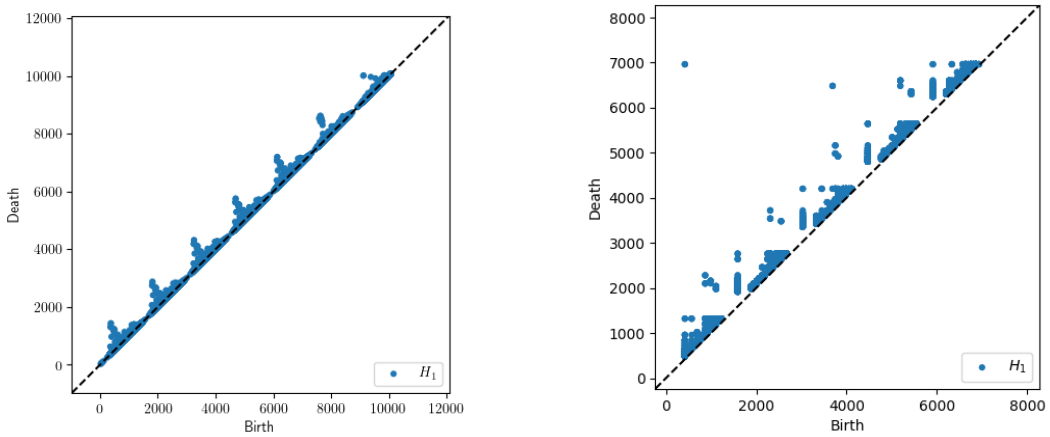

Figure 13: H_1 zigzag persistence diagram for the Great Britain Transportation Network (left) over a 7 day period and the learned embeddings over the training data which consists of the first 5 days. Our model did introduce some cycles which persisted across the 5 day training period which could correspond to the embeddings of groups of stations which constantly run in small periodic cycles being too close together. Despite there being features that differ from the ground truth zigzag persistence, our model was still able to recover many of the smaller periodic cycles corresponding to each day.

Algorithm 1 NEGATIVESAMPLING(data, e_ℓ, n_ℓ, sampling_rate)

Require: data $\in \mathbb{R}^{N\times(3+2n_\ell+e_\ell)}$ with columns $(u, v, t, \phi_u, \phi_v, \psi_{uv})$
Require: e_ℓ (edge feature length), n_ℓ (node feature length), sampling_rate $\in (0, 1]$
Ensure: (data, negatives) where negatives has $N_{\text{neg}} = \lfloor N \cdot \text{sampling\_rate} \rfloor$ rows

$N_{\text{neg}} \leftarrow \lfloor N \cdot \text{sampling\_rate} \rfloor$; $C \leftarrow 3 + 2n_\ell + e_\ell$
$S^+ \leftarrow \{(u_i, v_i, t_i)\}_{i=1}^{N}$ extracted from the first 3 columns of data
$\mathcal{T} \leftarrow$ unique times from the t column; $\mathcal{V} \leftarrow$ unique nodes from u and v columns
if $N_{\text{neg}} = 0$ **then**
    return (data, empty matrix of shape $0 \times C$)
end if
negatives $\leftarrow$ zero-initialized array of shape $N_{\text{neg}} \times C$
$c \leftarrow 0$
while $c < N_{\text{neg}}$ **do**
    sample distinct $u, v \sim \mathcal{V}$ without replacement; sample $t \sim \mathcal{T}$
    if $(u, v, t) \notin S^+$ **and** $(v, u, t) \notin S^+$ **then**
        $r_u \leftarrow$ first index r s.t. (data$[r, 0] = u$) **or** (data$[r, 1] = u$)
        $r_v \leftarrow$ first index r s.t. (data$[r, 0] = v$) **or** (data$[r, 1] = v$) ▷ find first rows containing u and v on either endpoint
        if data$[r_u, 0] = u$ **then**
            $\phi_u \leftarrow$ data$[r_u,\ 3 : 3 + n_\ell]$
        else
            $\phi_u \leftarrow$ data$[r_u,\ 3 + n_\ell : 3 + 2n_\ell]$
        end if
        if data$[r_v, 0] = v$ **then**
            $\phi_v \leftarrow$ data$[r_v,\ 3 : 3 + n_\ell]$
        else
            $\phi_v \leftarrow$ data$[r_v,\ 3 + n_\ell : 3 + 2n_\ell]$
        end if
        negatives$[c, :] \leftarrow (u, v, t, \phi_u, \phi_v, \mathbf{0}_{e_\ell})$ ▷ assemble a negative row with zeroed edge features
        $c \leftarrow c + 1$
    end if
end while
Sort rows of negatives by the 3rd column (time t) in ascending order.
return (data, negatives)

Algorithm 2 NODESEQUENCING(data, n_ℓ, e_ℓ)

Require: data $\in \mathbb{R}^{N\times(3+2n_\ell+e_\ell)}$ with columns $(u, v, t, \phi_u, \phi_v, \psi_{uv})$
Require: n_ℓ (node feature length), e_ℓ (edge feature length)
Ensure: padded_data (batched, time-ordered sequences per node), sort_indices, group_lengths

Column offsets
u_start $\leftarrow 3$; v_start $\leftarrow$ u_start $+ n_\ell$; edge_start $\leftarrow$ v_start $+ n_\ell$
Slice core columns
$u \leftarrow$ data$[:, 0]$; $v \leftarrow$ data$[:, 1]$; $t \leftarrow$ data$[:, 2]$
Slice feature blocks
$\phi_u \leftarrow$ data[:, u_start : u_start $+ n_\ell$]
$\phi_v \leftarrow$ data[:, v_start : v_start $+ n_\ell$]
$\psi_{uv} \leftarrow$ data[:, edge_start : edge_start $+ e_\ell$]
Build per-endpoint vectors
u-vectors $\leftarrow \mathrm{HorizontalStack}\big(\mathrm{Column}(u),\ \mathrm{Column}(t),\ \phi_u,\ \psi_{uv}\big)$
v-vectors $\leftarrow \mathrm{HorizontalStack}\big(\mathrm{Column}(v),\ \mathrm{Column}(t),\ \phi_v,\ \psi_{uv}\big)$
Interleave and sort by (node, time)
$D \leftarrow 2 + n_\ell + e_\ell$ ▷ columns: nodes, time, node/edge features
all_vectors $\leftarrow$ zeros of shape $(2N) \times D$
fill even rows with u-vectors and odd rows with v-vectors
sort_indices $\leftarrow \mathrm{LexSort}\big(\text{key}_1 = \text{all\_vectors}[:, 1],\ \text{key}_0 = \text{all\_vectors}[:, 0]\big)$
sorted_all $\leftarrow$ all_vectors[sort_indices]
Group by node and form sequences
initialize map $\mathcal{G}$ from node $\rightarrow$ list of vectors
for each vec in rows of sorted_all **do**
    node $\leftarrow$ vec[0]
    if node $\notin \mathcal{G}$ then create empty list
    append vec to $\mathcal{G}$[node]
end for
initialize lists group_tensors and group_lengths
for node in $\mathrm{sorted\_keys}(\mathcal{G})$ **do**
    arr $\leftarrow \mathrm{Stack}(\mathcal{G}[\text{node}])$
    ten $\leftarrow \mathrm{TorchTensor}(\text{arr})$
    append ten to group_tensors; append len(arr) to group_lengths
end for
Pad to batch tensor
padded_data $\leftarrow \mathrm{PadSequence}(\text{group\_tensors},\ \text{batch\_first} = \text{True},\ \text{padding\_value} = 0.0)$
return (padded_data, sort_indices, group_lengths)

Algorithm 3 PREPROCESSING

Require: temporal edge data data; node feature length n_ℓ; edge feature length e_ℓ; sampling rate ρ; model hyperparameters

Ensure: processed train/val/test sets with positive/negative samples, labels, and padded group tensors
Note: If using a TGBL dataset, adopt their *train/validation/test* splits directly; otherwise use chronological 70/15/15 splits.

$N \leftarrow \mathrm{Len}(\text{data})$
$N_{\text{train}} \leftarrow \lfloor 0.7\,N \rfloor, \quad N_{\text{val}} \leftarrow \lfloor 0.15\,N \rfloor, \quad N_{\text{test}} \leftarrow N - N_{\text{train}} - N_{\text{val}}$
pos_train $\leftarrow$ data$[0{:}N_{\text{train}}]$
pos_val $\leftarrow$ data$[N_{\text{train}}{:}N_{\text{train}} + N_{\text{val}}]$
pos_test $\leftarrow$ data$[N_{\text{train}} + N_{\text{val}}{:}]$
for split $\in$ {train, val, test} **do**
    (pos_split, neg_split) $\leftarrow \mathrm{NegativeSampling}(\text{pos\_split}, e_\ell, n_\ell, \rho)$
end for
train_data $\leftarrow \mathrm{Concatenate}(\text{pos\_train}, \text{neg\_train})$
val_data $\leftarrow \mathrm{Concatenate}(\text{pos\_val}, \text{neg\_val})$
test_data $\leftarrow \mathrm{Concatenate}(\text{pos\_test}, \text{neg\_test})$
train_labels $\leftarrow \mathrm{Tensor}\big(\mathrm{GetLabels}(\text{pos\_train}, \text{neg\_train})\big)$ ▷ 1 for positives, 0 for negatives
val_labels $\leftarrow \mathrm{Tensor}\big(\mathrm{GetLabels}(\text{pos\_val}, \text{neg\_val})\big)$
test_labels $\leftarrow \mathrm{Tensor}\big(\mathrm{GetLabels}(\text{pos\_test}, \text{neg\_test})\big)$
input_size $\leftarrow 2 + n_\ell$ ▷ node id + timestamp + node features
feature_size $\leftarrow e_\ell$ ▷ edge feature length
(padded_train, sort_train, len_train) $\leftarrow \mathrm{NodeSequencing}(\text{train\_data}, n_\ell, e_\ell)$
(padded_val, sort_val, len_val) $\leftarrow \mathrm{NodeSequencing}(\text{val\_data}, n_\ell, e_\ell)$
(padded_test, sort_test, len_test) $\leftarrow \mathrm{NodeSequencing}(\text{test\_data}, n_\ell, e_\ell)$
return training/validation/test tensors, sort indices, labels, and group lengths

Algorithm 4 COMPUTEDISKLOSS(embeddings, ground_truth, sort_indices, group_lengths)

Require: embeddings; ground_truth labels; sort_indices; group_lengths; temporal model
Ensure: binary cross-entropy loss

Recover the edge orderings prior to sequencing.

recovered_vectors $\leftarrow [\,]$ ▷ empty list
$start\_idx \leftarrow 0$
for each $length$ in group_lengths **do**
    $group\_vectors \leftarrow$ embeddings$[start_idx][0 : length]$ ▷ Pulls embedding for the node id, ignoring padded entries
    append $group\_vectors$ to recovered_vectors
    $start\_idx \leftarrow start\_idx + 1$
end for
recovered_sorted_all $\leftarrow \mathrm{Concatenate}$(recovered_vectors, dim $= 0$)
reverted_array $\leftarrow$ EmptyLike(recovered_sorted_all)
reverted_array[sort_indices] $\leftarrow$ recovered_sorted_all ▷ Uses the sort indices generated by Algorithm 3

Compute edge probabilities

probs $\leftarrow$ COMPUTEPROBABILITIES(reverted_array) ▷ shape $(N,)$, from Equation C.3

Binary cross-entropy loss

$gt \leftarrow$ ground_truth_labels ▷ Uses the ground truth labels generated by Algorithm 3
loss $\leftarrow \mathrm{BCE}$(probs, gt)
return loss