Temporal Graph Learning

JULY 2024

Static/dynamic graph Formulizations

Definition 1 (Static Graph - SG) *A* Static Graph *is a tuple* $G = (V, E, X^V, X^E)$, *where* V *is the set of nodes,* $E \subseteq V \times V$ *is the set of edges, and* X^V, X^E *are* d_V*-dimensional node features and* d_E*-dimensional edge features.*

Definition 2 (Temporal Graph - TG) *A* Temporal Graph *is a tuple* $G_T = (V, E, V_T, E_T)$, *where* V *and* E *are, respectively, the set of all possible nodes and edges appearing in a graph at any time, while*

$$V_T := \{(v, x^v, t_s, t_e) : v \in V, x^v \in \mathbb{R}^{d_V}, t_s \leq t_e\},$$
$$E_T := \{(e, x^e, t_s, t_e) : e \in E, x^e \in \mathbb{R}^{d_E}, t_s \leq t_e\},$$

are the temporal nodes and edges, with time-dependent features and initial and final timestamps. A set of temporal graphs is denoted as $\mathcal{G}_\mathcal{T}$.

Definition 3 (Discrete Time Temporal Graph - DTTG) *Let* $\Delta t > 0$ *be a fixed time-step and let* $t_1 < t_2 < \cdots < t_n$ *be timestamps with* $t_{k+1} = t_k + \Delta t$. *A* Discrete Time Temporal Graph G_{DT} *is a TG where for each* $(v, x^v, t_s, t_e) \in V_T$ *or* $(e, x^e, t_s, t_e) \in E_T$, *the timestamps* t_s, t_e *are taken from the set of fixed timestamps (i.e.,* $t_s, t_e \in \{t_1, t_2, \ldots, t_n\}$, *with* $t_s < t_e$*).*

Definition 4 (Snapshot-based Temporal Graph - STG) *Let* $t_1 < t_2 < \cdots < t_n$ *be the ordered set of all timestamps* t_s, t_e *occurring in a TG* G_T. *Set*

$$V_i := \{(v, x^v) : (v, x^v, t_s, t_e) \in V_T, t_s \leq t_i \leq t_e\},$$
$$E_i := \{(e, x^e) : (e, x^e, t_s, t_e) \in E_T, t_s \leq t_i \leq t_e\},$$

and define the snapshots $G_i := (V_i, E_i)$, $i = 1, \ldots, n$. *Then a* Snapshot-based Temporal Graph *representation of* G_T *is the sequence*

$$G_T^S := \{(G_i, t_i) : i = 1, \ldots, n\}$$

of time-stamped static graphs.

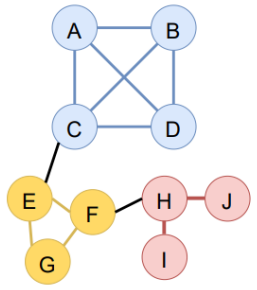
Definition 5 (Event-based Temporal Graph - ETG) *Let* G_T *be a TG, and let* ε *denote one of the following* events*:*

- Node insertion $\varepsilon_V^+ := (v, t)$*: the node* v *is added to* G_T *at time* t*, i.e., there exists* $(v, x^v, t_s, t_e) \in V_T$ *with* $t_s = t$.
- Node deletion $\varepsilon_V^- := (v, t)$*: the node* v *is removed from* G_T *at time* t*, i.e., there exists* $(v, x^v, t_s, t_e) \in V_T$ *with* $t_e = t$.
- Edge insertion $\varepsilon_E^+ := (e, t)$*: the edge* e *is added to* G_T *at time* t*, i.e., there exists* $(e, x^e, t_s, t_e) \in E_T$ *with* $t_s = t$.
- Edge deletion $\varepsilon_E^- := (e, t)$*: the edge* e *is removed from* G_T *at time* t*, i.e., there exists* $(e, x^e, t_s, t_e) \in E_T$ *with* $t_e = t$.

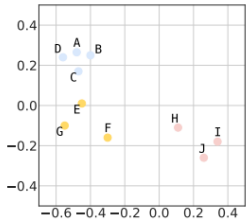
An Event-based Temporal Graph *representation of TG is a sequence of events*

$$G_T^E := \{\varepsilon : \varepsilon \in \{\varepsilon_V^+, \varepsilon_V^-, \varepsilon_E^+, \varepsilon_E^-\}\}.$$

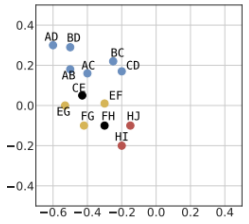
Graph Embedding



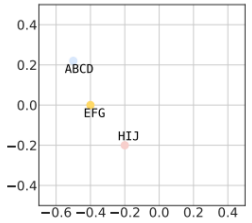
(a) Graph G



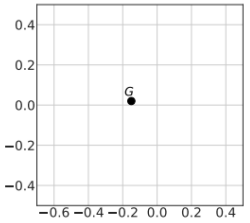
(b) Node Embedding



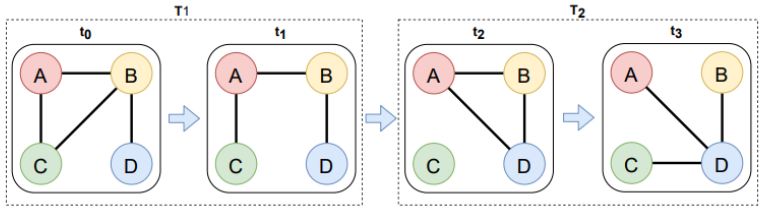
(c) Edge Embedding



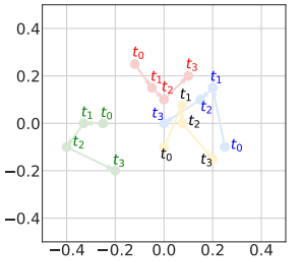
(d) Substructure Embedding



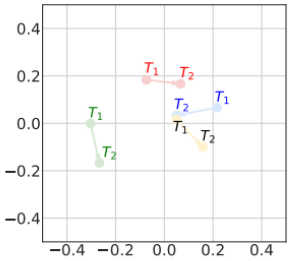
(e) Whole-Graph Embedding



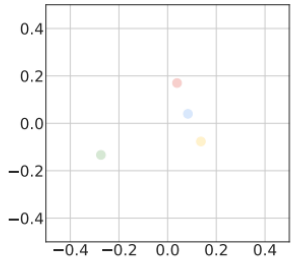
(a) Dynamic graph $\mathcal{G}$



(b) Embedding dynamic graph

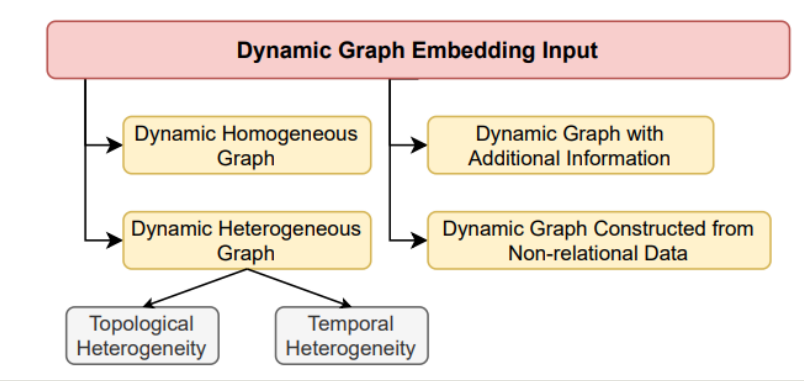


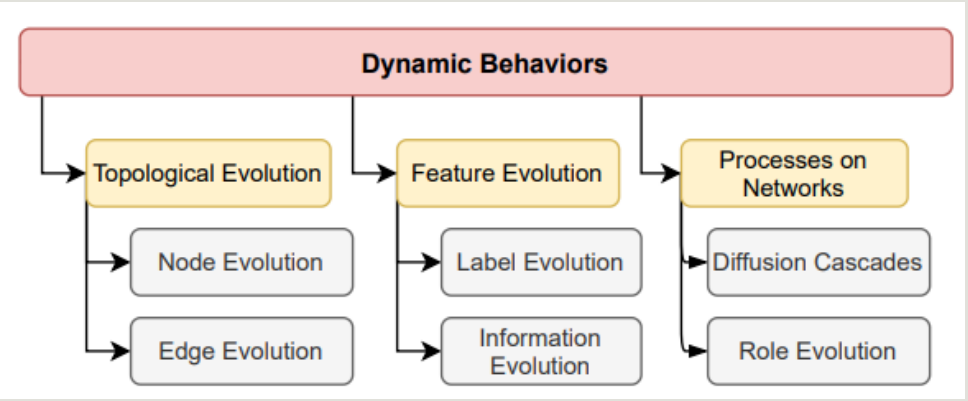
(c) Time-grouping embedding

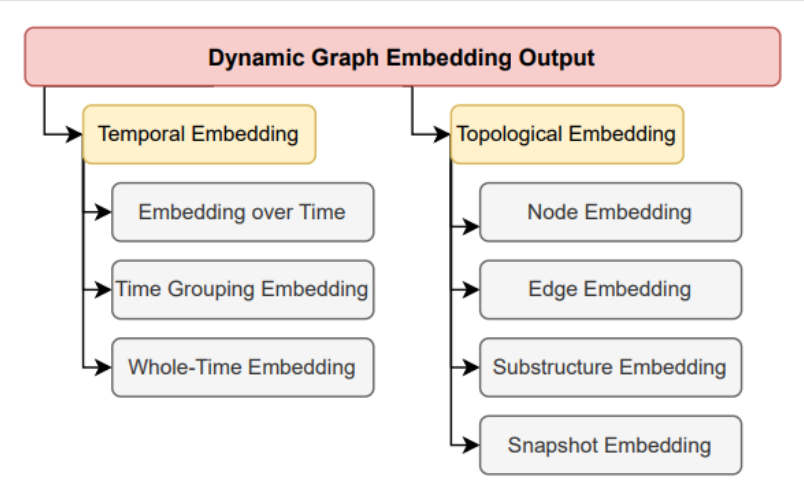


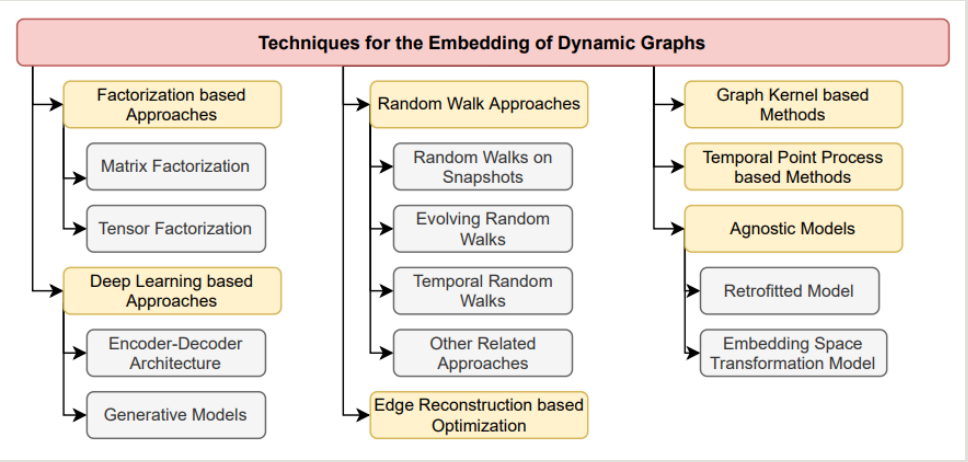
(d) Whole-time embedding

Dynamic Graph Embedding

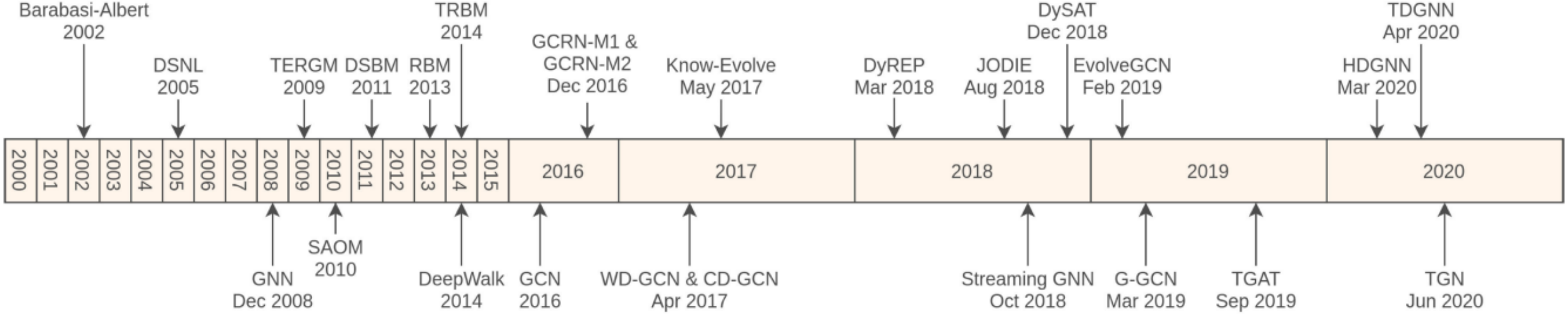








Timeline of dynamic graph neural networks



Example DGNN Embedding methods

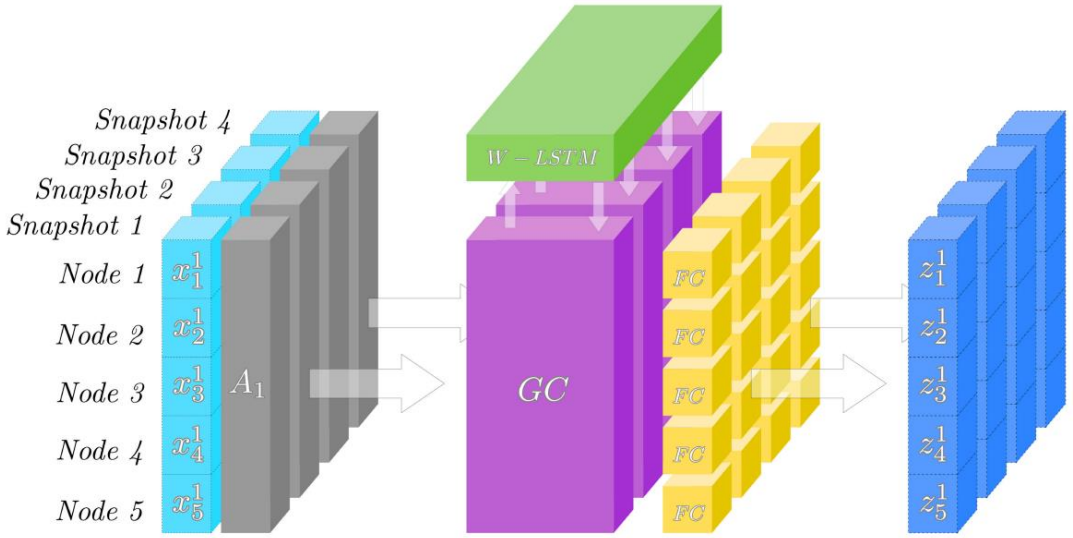


FIGURE 9. Integrated DGNN structure of EvolveGCN with an EGCU-O layer [16]. The EGCU-O layer constitutes the GC (graph convolution) and the W-LSTM (LSTM for GC weights). W-LSTM is used to initialize the weights of the GC.

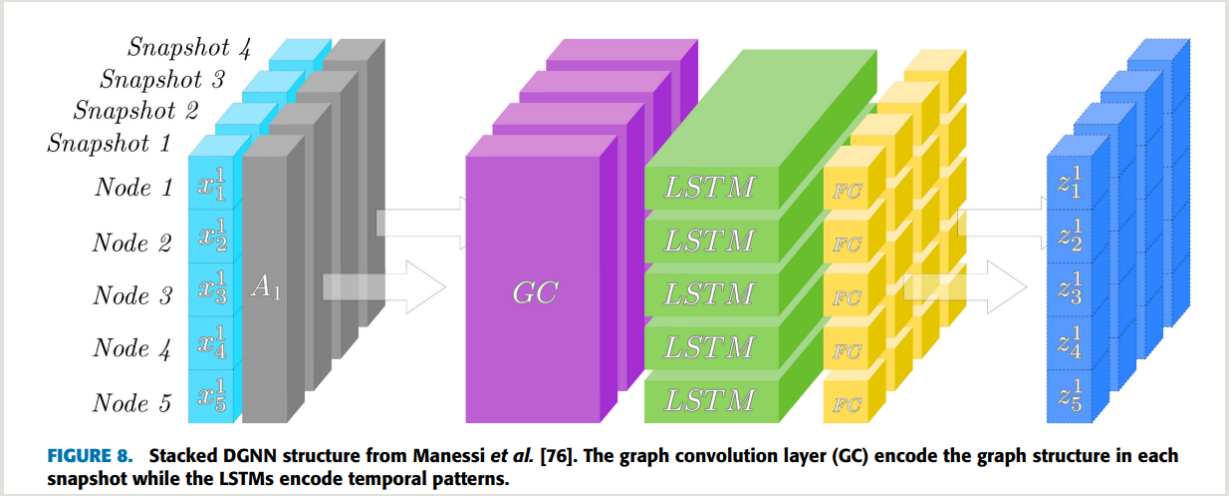


FIGURE 8. Stacked DGNN structure from Manessi *et al.* [76]. The graph convolution layer (GC) encode the graph structure in each snapshot while the LSTMs encode temporal patterns.

Node-based and edge-based

- Node-based models: First leverage the node information such as temporal neighborhood or previous node history to generate node embeddings and then aggregate node embeddings from both source and destination node of an edge to predict its existence.

- TGN, DyRep, TCL, …

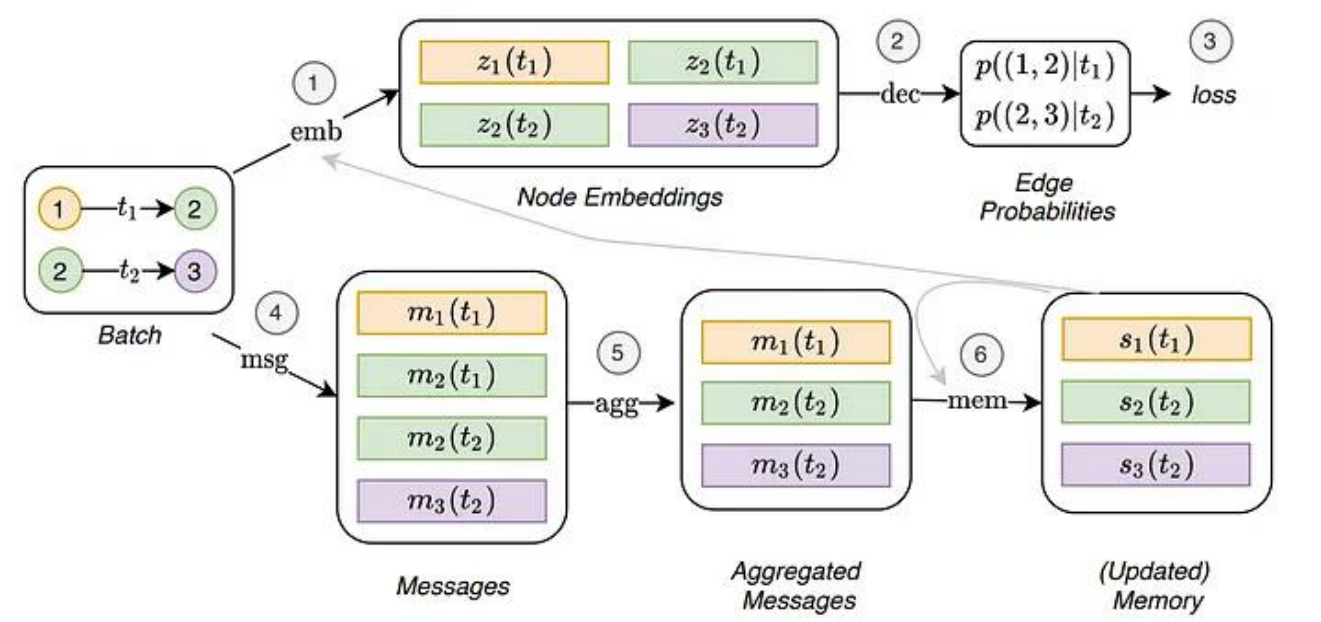
- Edge-based methods: Aim to directly generate embeddings for the edge of interest and then predict its existence.

- CAWN [46] and GraphMixer [9]

TGNs

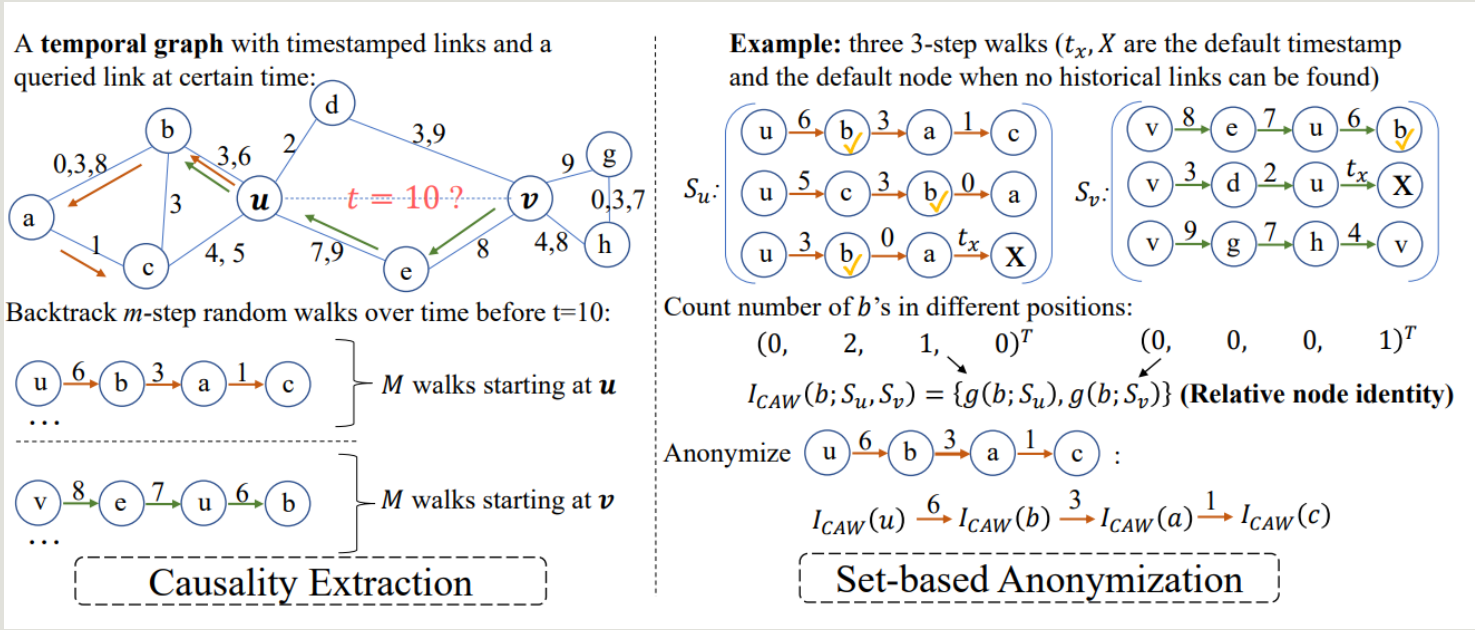
- Generalize Message Passing Neural Networks (MPNNs) to temporal graphs.
- Generalizes Joint Dynamic User-Item Embeddings (JODIE) and Temporal Graph Attention (TGAT)

- **Node memory** which represents the state of the node at a given time, acting as a compressed representation of the node's past interactions.



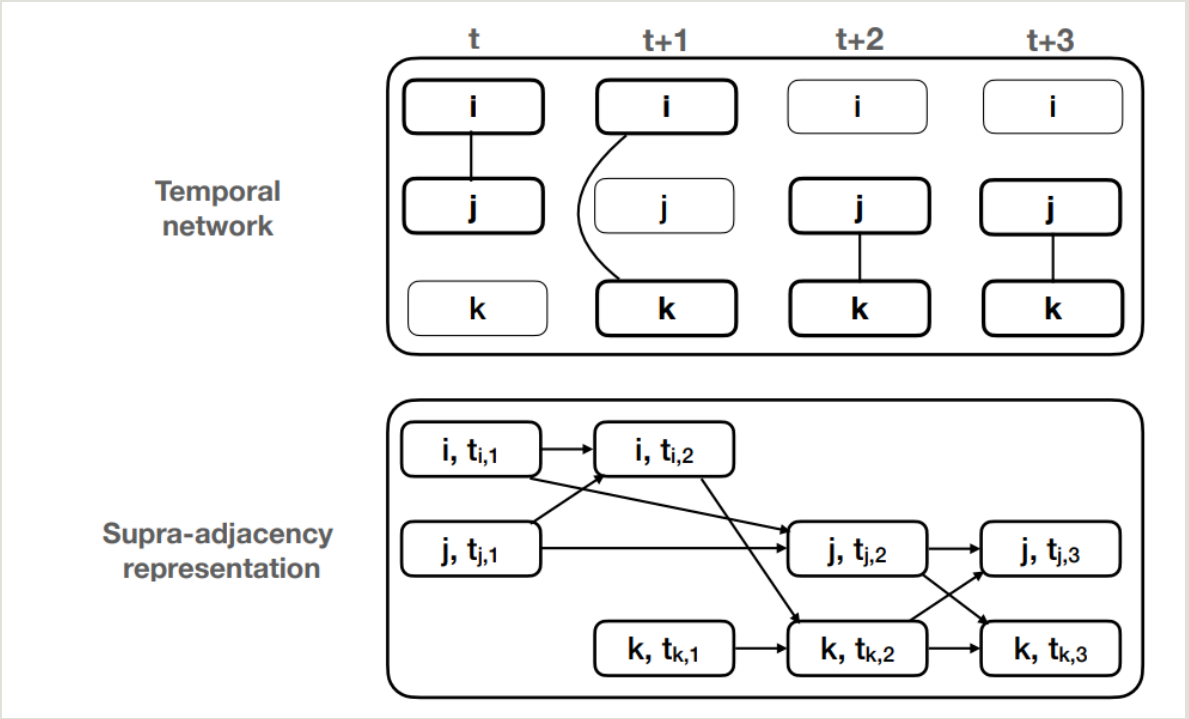
Walk Aggregating methods

- They rely on temporal random walks.
- Each walk is encoded using an RNN,
- Encodings are then aggregated by using self-attention or taking a simple average.



DyANE

- The temporal graphs -> static graph representation (supra-adjacency representation)



DyRep

Learns a set of functions that can effectively generate evolving, lowdimensional node embeddings. Then a temporal point process is employed to estimate the likelihood of an edge between nodes at a certain timestamp.

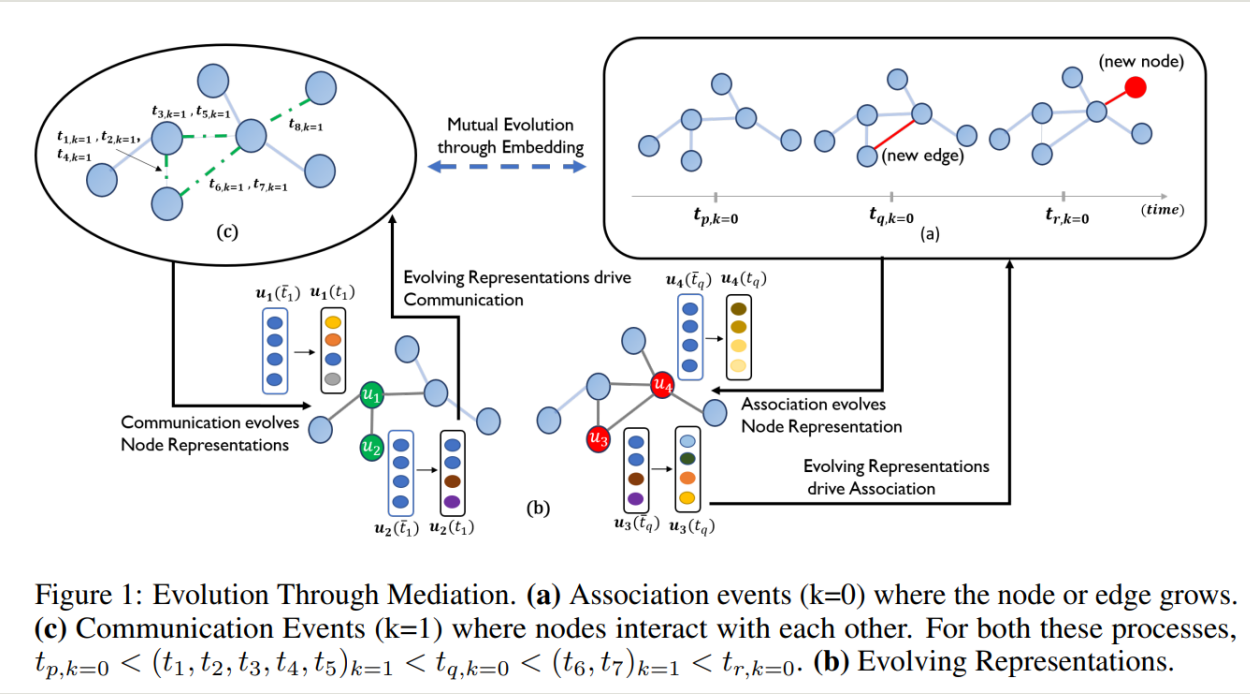


Figure 1: Evolution Through Mediation. **(a)** Association events (k=0) where the node or edge grows. **(c)** Communication Events (k=1) where nodes interact with each other. For both these processes, $t_{p,k=0} < (t_1, t_2, t_3, t_4, t_5)_{k=1} < t_{q,k=0} < (t_6, t_7)_{k=1} < t_{r,k=0}$. **(b)** Evolving Representations.

DynGem

DynGem is a dynamical autoencoder for growing graphs that construct the embedding of a snapshot based on the embedding of the previous snapshot.

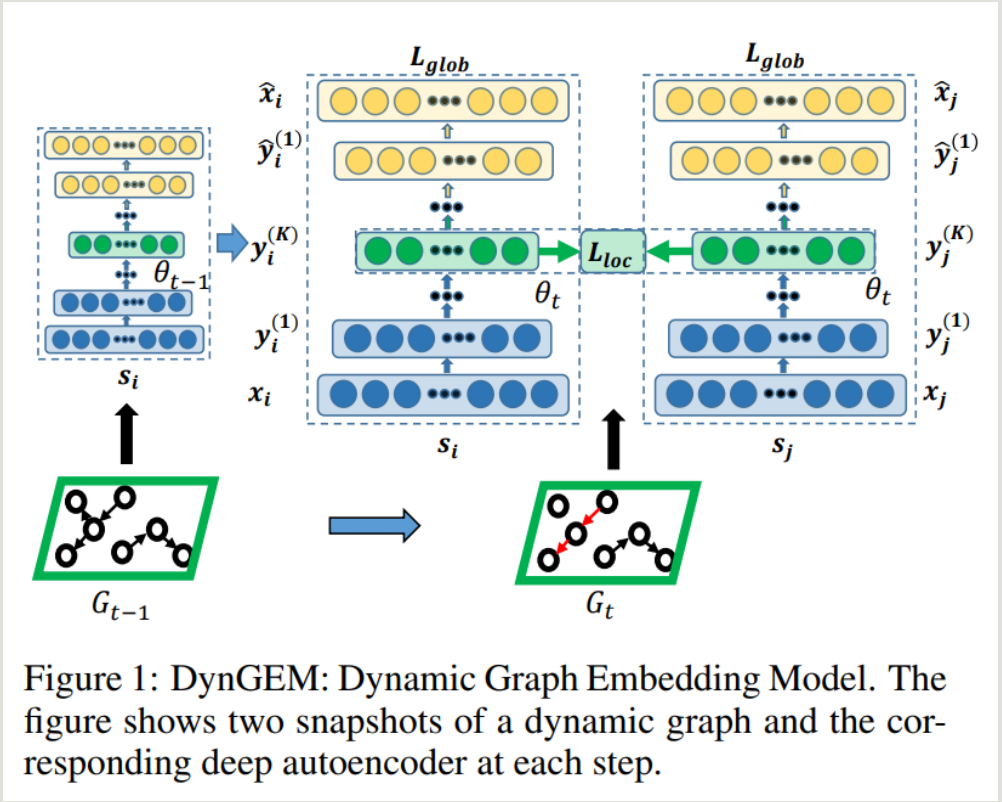
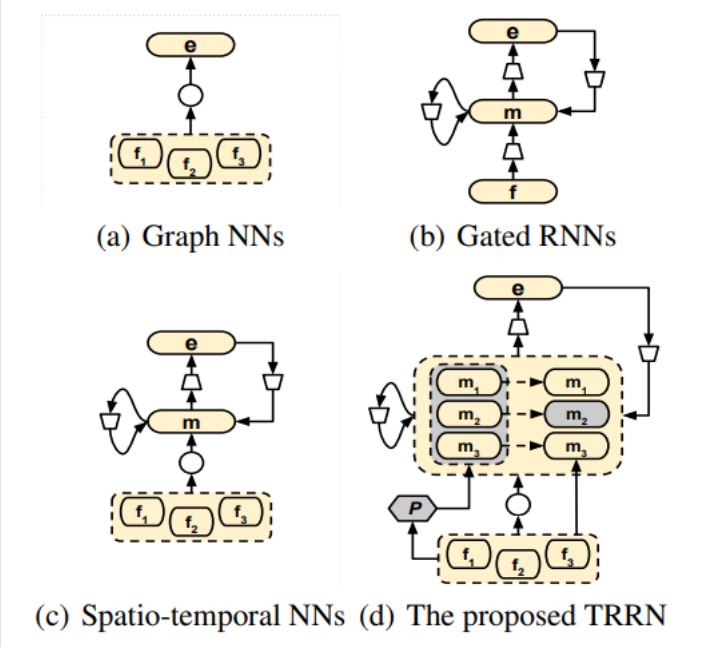


Figure 1: DynGEM: Dynamic Graph Embedding Model. The figure shows two snapshots of a dynamic graph and the corresponding deep autoencoder at each step.

TRRN

Uses multi-head self-attention to process a set of memories,

enabling efficient information flow from past observations to current latent representations through shortcut paths. It incorporates policy networks with differentiable binary routers to estimate the activation probability of each memory and dynamically update them at the most relevant time steps.



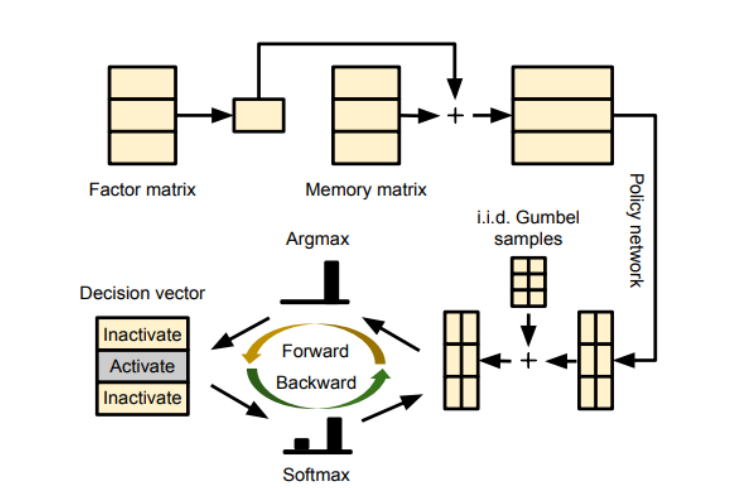


Figure 2: Illustration of activating memories selectively.

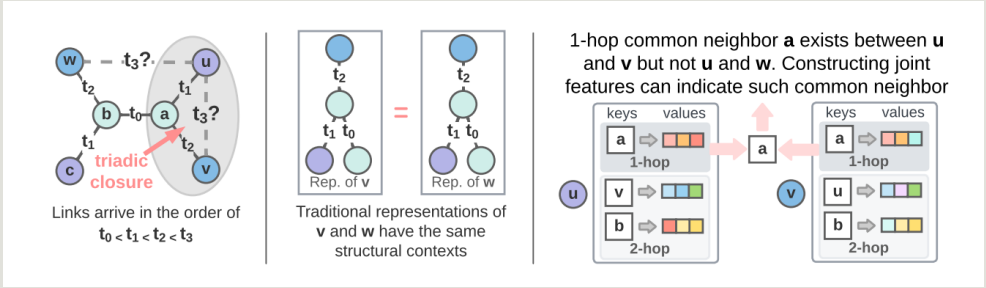
Neighborhood-Aware Temporal network model

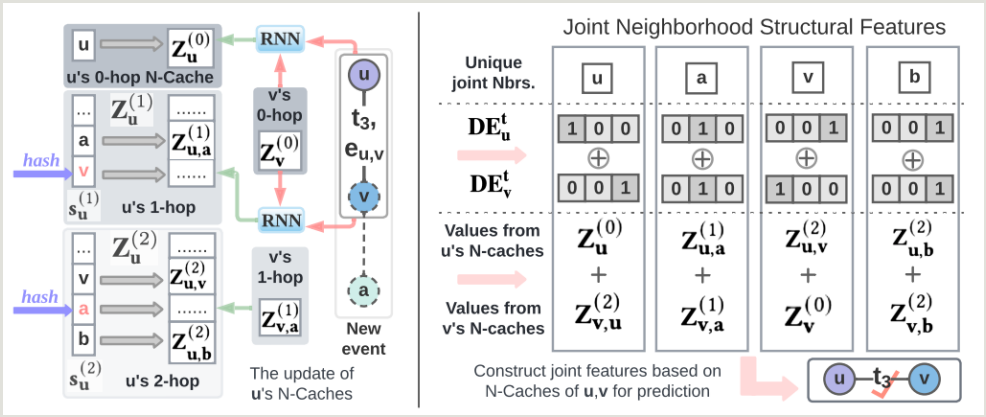
explicitly model the joint neighborhood of a set of nodes for future **link prediction**.

The joint neighborhood is not captured by traditional Graph Neural Network (GNN) based approaches as the node embedding vectors are generated independently for each node.

NAT adapted a novel dictionary-type neighborhood representation which records k-hop neighborhood information and allows fast construction of structure features of joint neighborhood of multiple nodes.

The dictionary representation is maintained by an efficient cache technique named N-cache. N-caches allowed NAT to construct the joint neighborhood features for a batch of node pairs for fast link prediction.





DyGFormer

Aims to capture long-term temporal dependencies by proposing DyGFormer, a new Transformer-based architecture for temporal graph learning.

The first step is to extract historical first-hop interactions of nodes. This includes the encodings of neighbors, links, time intervals as well as the frequencies of every neighbor's appearances.

The assumption is that if two nodes share more common historical neighbors in the past, then they are more likely to interact in the future. After encoding the historical interactions in a sequence, it is then divided into multiple patches and fed into a transformer for capturing temporal dependencies.

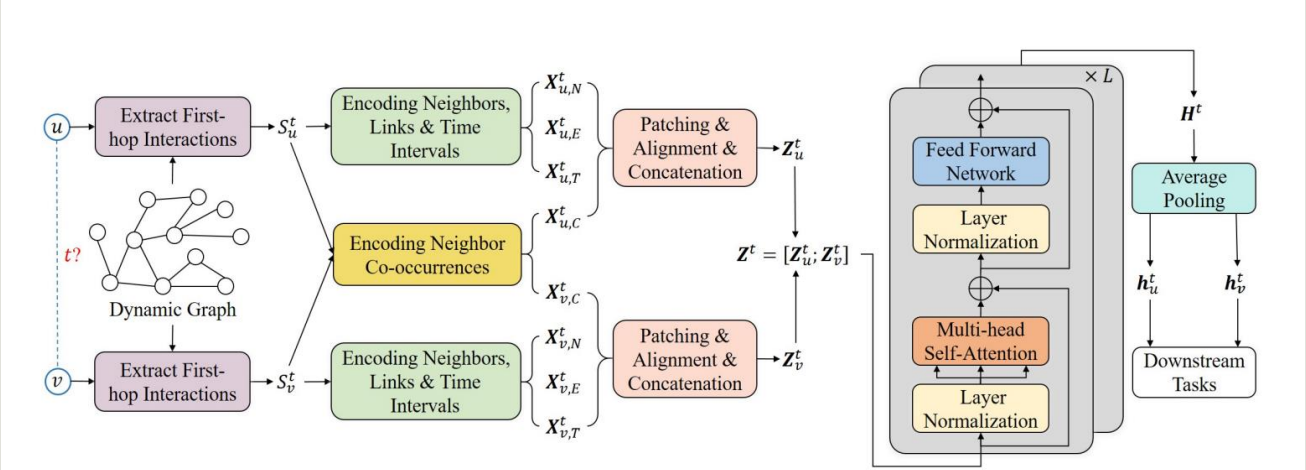


Figure 1: Framework of the proposed model.

GraphMixer

The model is based entirely on MLPs and neighbor mean-pooling while performing strongly against baselines with RNN and self-attention.

GraphMixer contains three modules:

- a link-encoder summarizes the information from temporal links,

- a node-encoder extracts information from nodes

- a link-classifier which combines the above information for prediction.

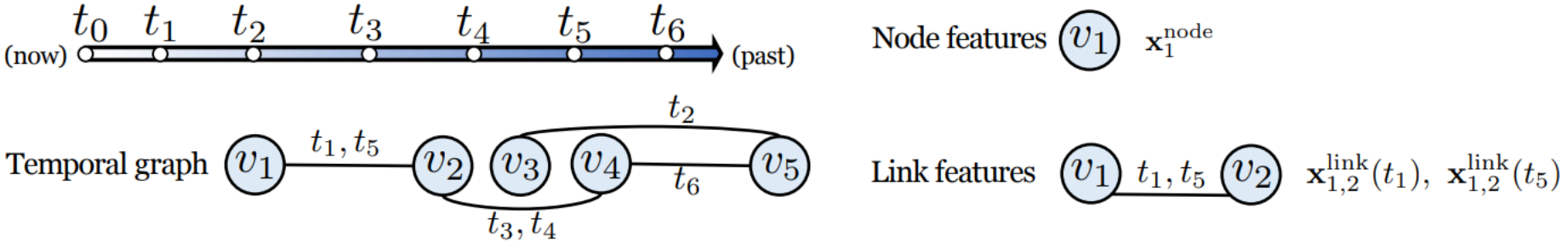
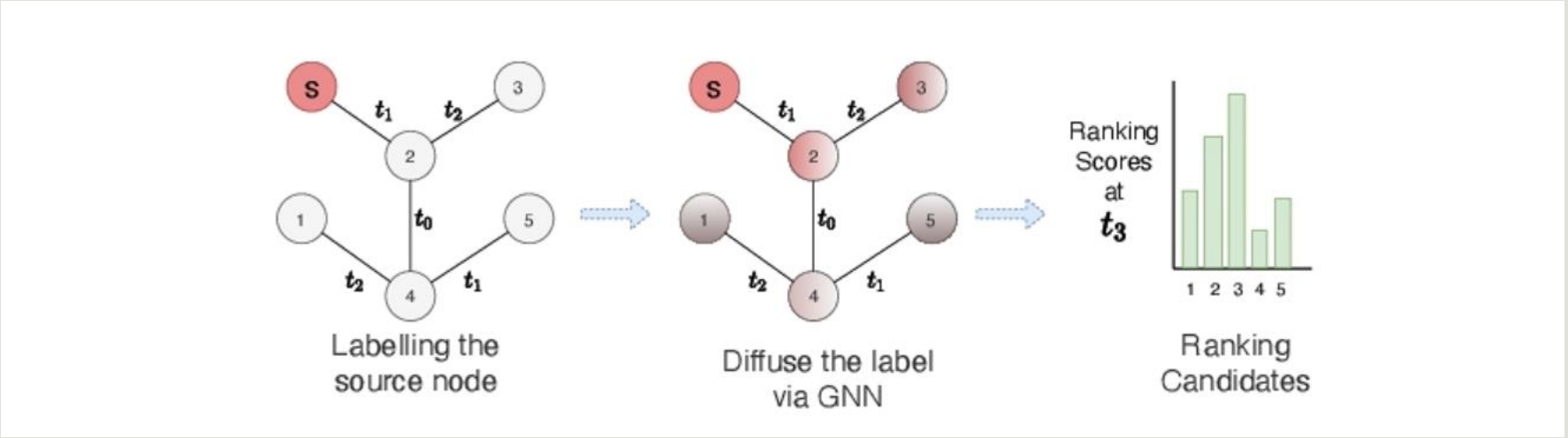


Figure 1: (Left) Temporal graph with nodes $v_1, \ldots, v_5$, per-link timestamps $t_1, \ldots, t_6$ indicate when two nodes interact. For example, v_1, v_2 interact at t_1, t_5. (Right) Each node has its node features (e.g., $\mathbf{x}_1^{\text{node}}$ for v_1) and each temporal link has its link features (e.g., $\mathbf{x}_{1,2}^{\text{link}}(t_1), \mathbf{x}_{1,2}^{\text{link}}(t_5)$ are link features between v_1, v_2 at t_1, t_5). For scenarios without node or link features, we use all-zero vectors instead.

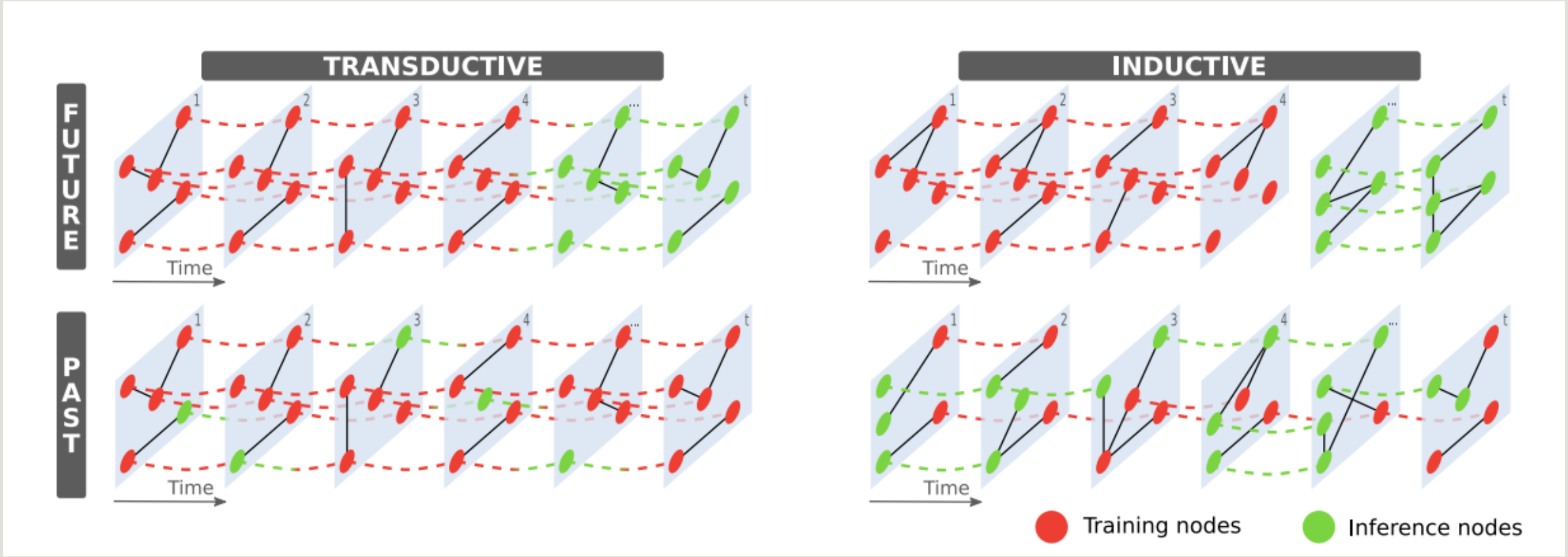
TGRank

Existing methods maximizes accuracy independently over future links, ignoring the fact that future links often have dependency between each other.

Therefore, Suresh et al. treat dynamic link prediction as a ranking problem and propose Temporal Graph network for RANKing (TGRank) to learn to rank over a list of candidates. The task query now contains a center node s with a set of candidate nodes (all other nodes in the subgraph) and the goal is to rank the most likely candidate as the destination of node s.



Learning settings



Supervised learning tasks

- Classification:

- Temporal Node Classification (maps each node to a class, at a time t)

- Temporal Edge Classification (which assigns each edge to a class at a given time t)

- Temporal Graph Classification (maps a temporal graph, restricted to a time interval [ts, te], into a class.)

- Regression (replacing the categorical target C with the set R)

- Link prediction:

- Temporal Link Prediction (predicts the probability that, at a certain time, there exists an edge between two given nodes.)

- Event Time Prediction (predicts the time of the first appearance of an edge)

Node affinity prediction

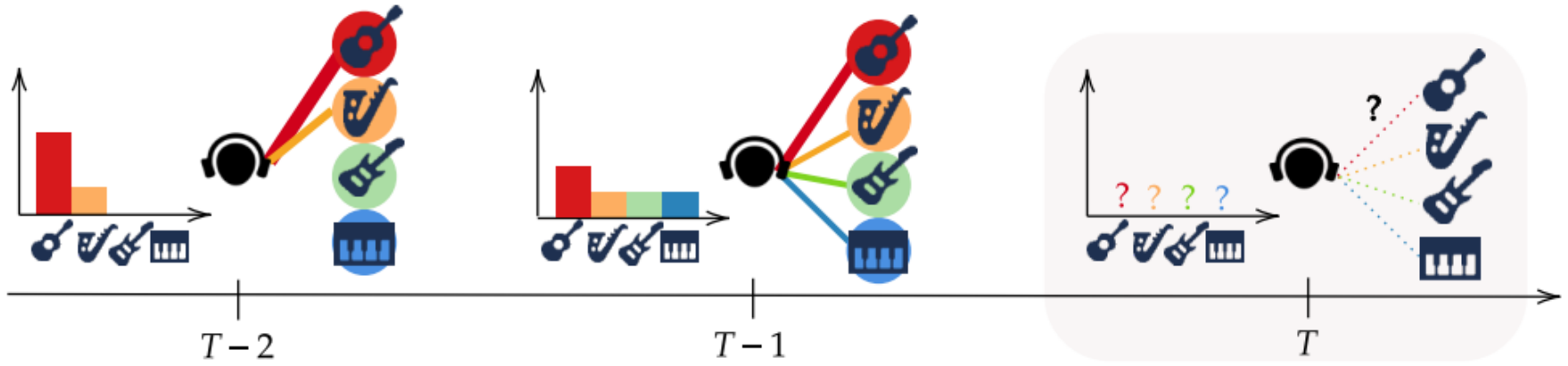
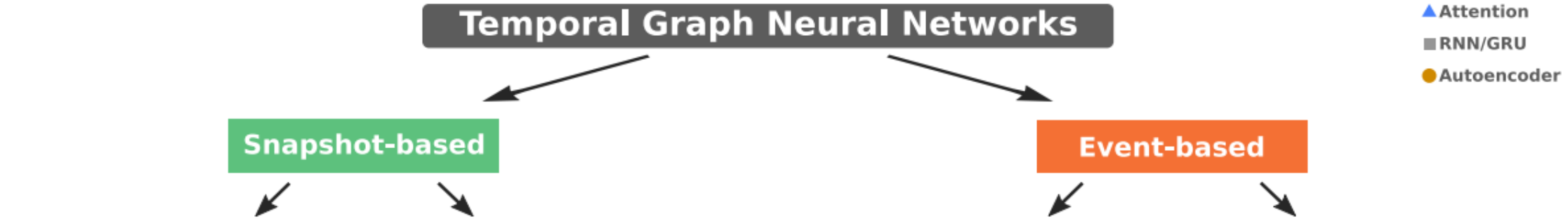


Figure 3: The *node affinity prediction* task aims to predict how the preference of a user towards items change over time. In the `tgbn-genre` example, the task is to predict the frequency at which the user would listen to each genre over the next week given their listening history until today.

Unsupervised learning tasks

- Clustering:
  - Temporal Node Clustering
  - Temporal Graph Clustering
- Low-dimensional embedding (LDE):
  - Low-dimensional temporal node embedding
  - Low-dimensional temporal graph embedding



	EvolveGCN	VGRNN	DySAT	DynGESN	Roland	SSGNN	TGAT	NAT	TGL	APAN	DGNN	TGN
f_{NC}	FT	✗	✗	✗	✗	✗	FT,FI	✗	FT	FT	PT	FT
f_{EC}	FT	✗	✗	✗	✗	✗	✗	✗	✗	FT	✗	✗
f_{GC}	✗	✗	✗	PT	✗	✗	✗	✗	✗	✗	✗	✗
f_{LP}	FT	FI,FT,PT	PT,FT	✗	FT	✗	FT,FI	PI,FT	FT	FT	FT	FI,FT
f_{EP}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Clustering	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
LDE	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Regression (node)	✗	✗	✗	FT	✗	FT	✗	✗	✗	✗	✗	✗

Explainable Temporal Graph Methods

T-GNNExplainer is model-agnostic and finds important events from a set of candidate events to best explain the model prediction. Xia et al. treat the problem of identifying a subset of explaining events as a combinatorial optimization problem by searching over a subset of the temporal graph within a given size. To tackle this, T-GNNExplainer employs an explorer-navigator framework. The navigator is trained from multiple target events to capture inductive correlations between events while the explorer searches out a specific combination of events based on Monte Carlo Tree Search, including node selection, node expansion, reward simulation and backprop. Which events are pruned is inferred from the navigator.

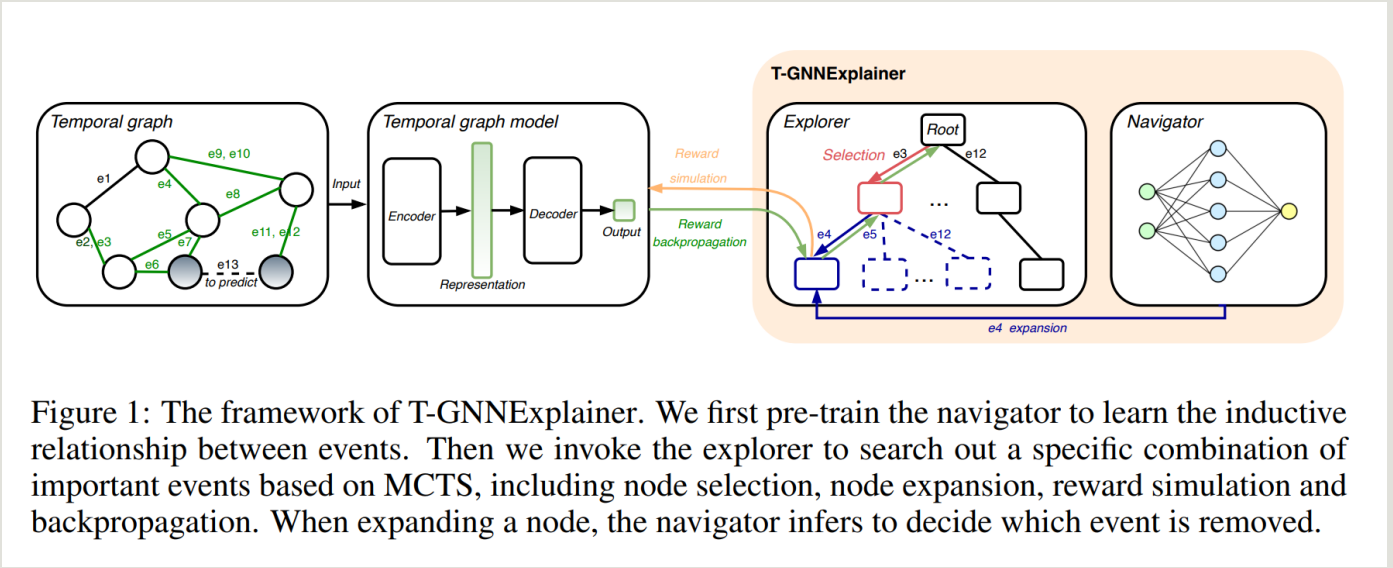


Figure 1: The framework of T-GNNExplainer. We first pre-train the navigator to learn the inductive relationship between events. Then we invoke the explorer to search out a specific combination of important events based on MCTS, including node selection, node expansion, reward simulation and backpropagation. When expanding a node, the navigator infers to decide which event is removed.

Rethinking Evaluation in Temporal Graphs

Evaluation on the link prediction task on dynamic graphs often involves:

1) fixed train, test split

2) random negative edge sampling and

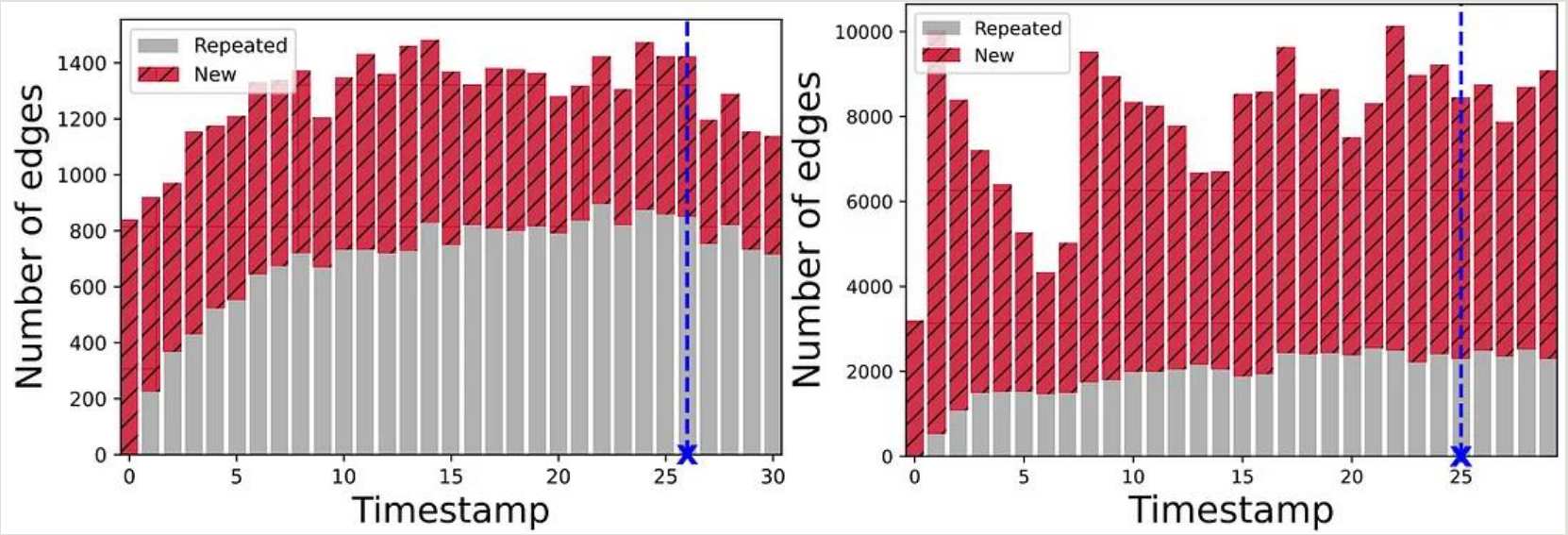
3) small datasets from similar domains

Such a fixed split means only edges from the chosen test period would be evaluated thus long term behavior potentially spanning training, validation and test period would not be correctly evaluated. In addition, many TGL methods are **stale** at test time, meaning that the model representation is not updated with information during evaluation.

Rethinking Evaluation in Temporal Graphs

Negative edges: 1) Easy: random negative edges

However, many edges in a temporal graph recur over time. Considering the sparsity of real world graphs, the majority of node pairs are unlikely to form an edge.

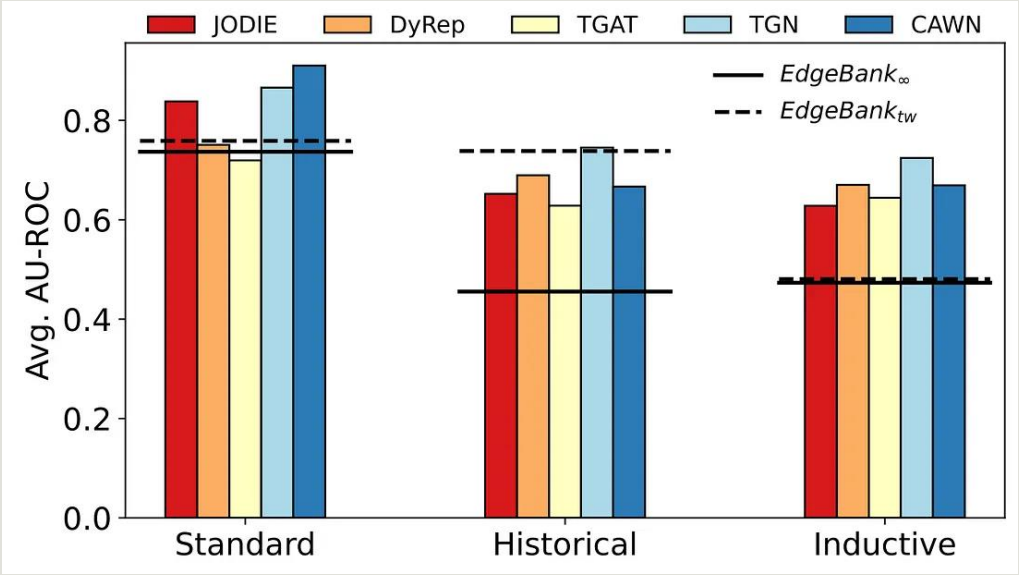


Rethinking Evaluation in Temporal Graphs

2) Historical negative edges: edges that appeared in the training set but are absent in the current test step.

3) Inductive negative edges as test edges which occurred previously in the test set but are not present at the current step.

A baseline EdgeBank, relying solely on memorizing past edges (essentially a hashtable of seen edges).



TGB Metrics

For the dynamic link prediction: AUROC or Average Precision

An appropriate metric should be able to capture the ranking of a positive edge amongst the negative ones

TGB Metrics

MRR: Mean Reciprocal Rank

The MRR computes the reciprocal rank of the true destination node among the negative or fake destinations.

The MRR varies in the range of (0, 1]

Commonly used metric in recommendation systems and knowledge graphs.

Perform collision checks to ensure that no positive edge is sampled as a negative edge.

Calculating MRR

1. **Generate predictions:** For a given actor and movie pair, our model predicts a probability score indicating the likelihood of the actor being in the movie.
2. **Rank predictions:** We rank all potential movies for that actor based on their predicted probabilities.
3. **Determine correct rank:** If the actual movie is in the top predicted list, we note its position (rank).
4. **Calculate reciprocal rank:** For each query (actor-movie pair), we calculate 1/rank.
5. **Average reciprocal ranks:** We compute the average of the reciprocal ranks across all queries.

TGB Metrics

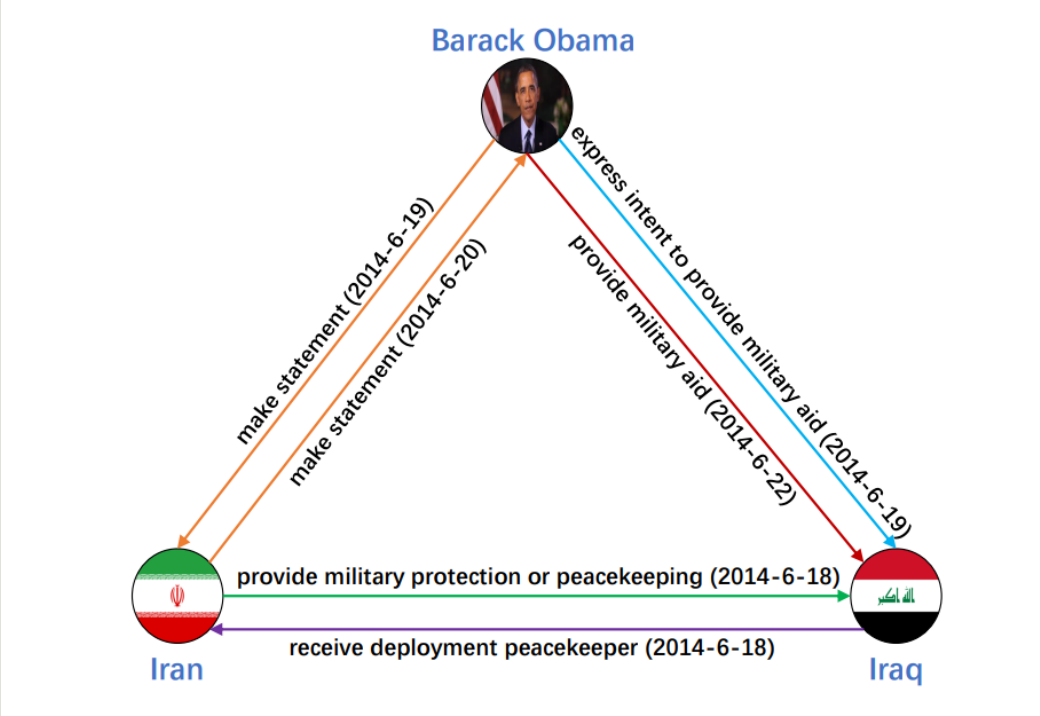
NDCG

https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0

$$\mathrm{nDCG_k} = \frac{\mathrm{DCG_k}}{\mathrm{IDCG_k}}$$

$$\mathrm{DCG_k} = \sum_{i=1}^{k} \frac{\mathrm{rel}_i}{\log_2(i+1)} \qquad \mathrm{IDCG_k} = \sum_{i=1}^{k} \frac{\mathrm{rel}_i}{\log_2(i+1)}$$

Temporal Knowledge Graphs



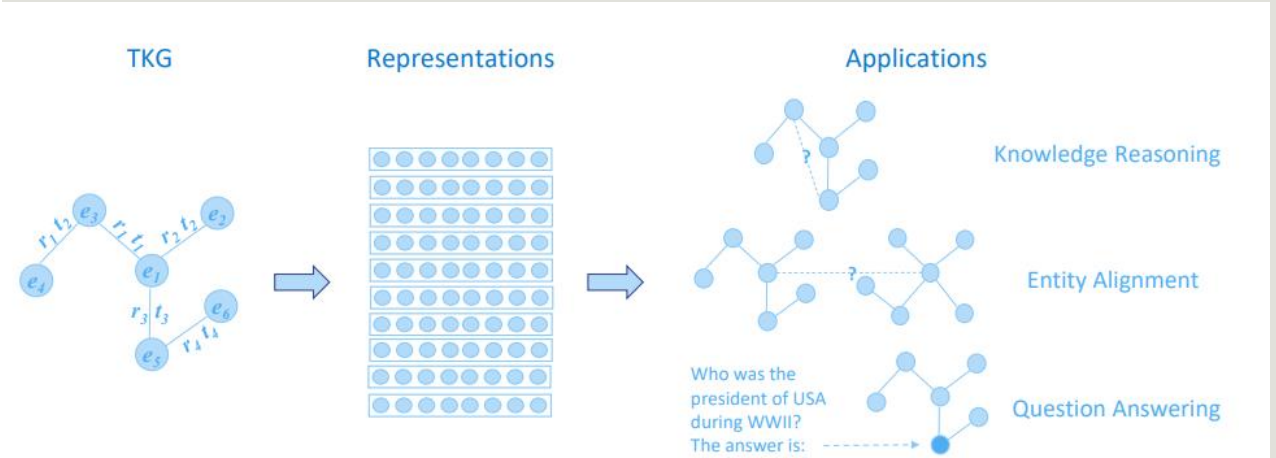


Figure 2: Temporal knowledge graph representation learning and applications.

Temporal Knowledge Graphs

Krause et al. has taken the first effort towards bridging the gap between temporal KGs and homogeneous graphs. In this work, the authors propose a framework to formalize various temporal aspects in KGs. Namely, they define temporal KGs as local extensions, i.e., graphs that have timestamps on the edges, and dynamic KGs as global extensions, i.e., graphs that change topology over time by adding or removing nodes and edges.

KNOWLEDGE GRAPH	+	LOCAL EXTENSION		
+		NO	YES	
GLOBAL EXTENSION	NO	STANDARD	REMINISCENT	STATIONARY
	YES	MUTABLE	INCREMENTAL	DYNAMIC
		STATIC	TEMPORAL	

Figure 1. Overview of the time-aware knowledge graph extensions regarding possible combinations of local extensions (edge timestamps) and global extensions (consideration of multiple consecutive versions of a graph).

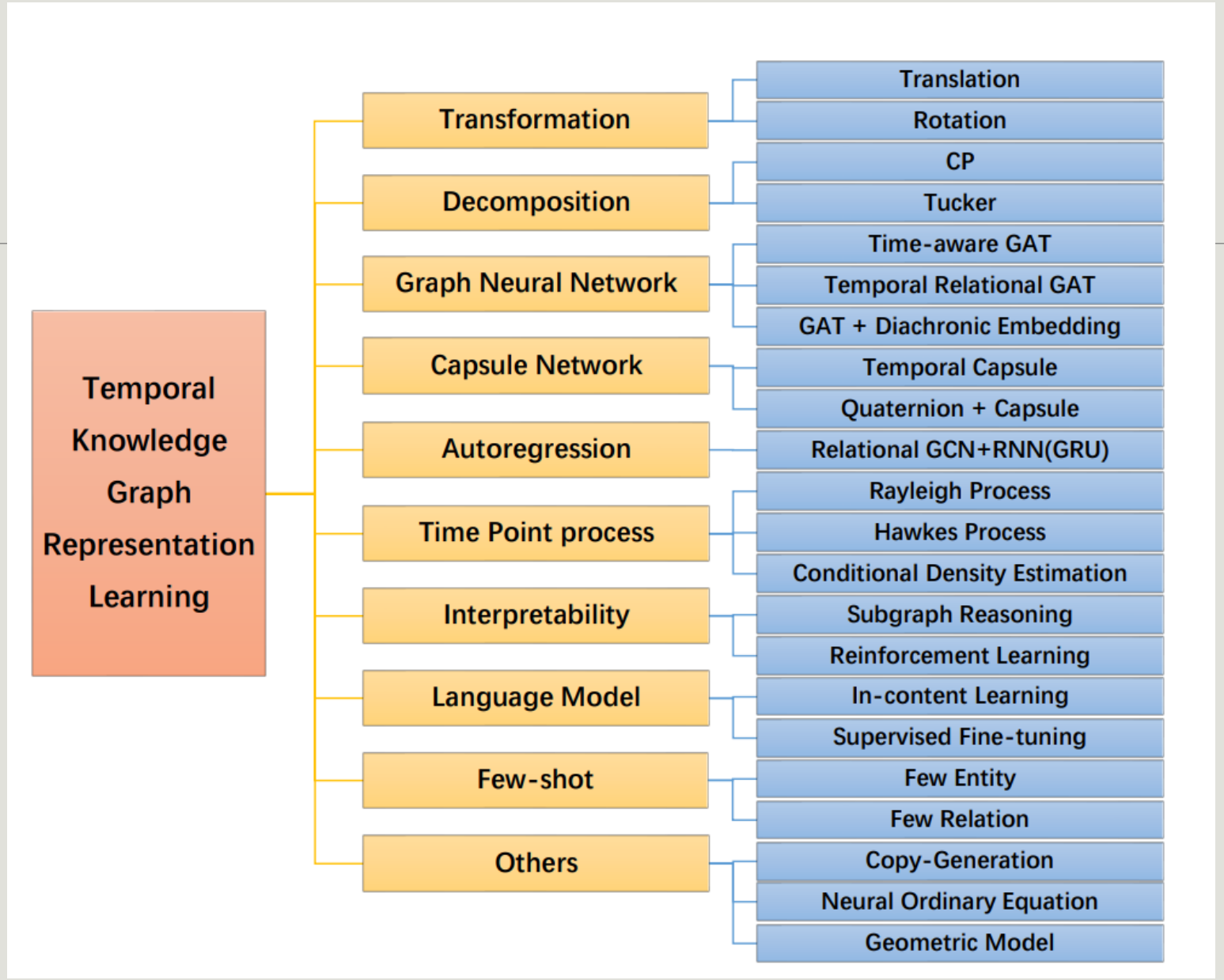

Temporal Knowledge Graph Representation Learning
Transformation
Translation
Rotation
Decomposition
CP
Tucker
Graph Neural Network
Time-aware GAT
Temporal Relational GAT
GAT + Diachronic Embedding
Capsule Network
Temporal Capsule
Quaternion + Capsule
Autoregression
Relational GCN+RNN(GRU)
Time Point process
Rayleigh Process
Hawkes Process
Conditional Density Estimation
Interpretability
Subgraph Reasoning
Reinforcement Learning
Language Model
In-content Learning
Supervised Fine-tuning
Few-shot
Few Entity
Few Relation
Others
Copy-Generation
Neural Ordinary Equation
Geometric Model

Table 3: A summary of recent TKGRL methods.

Category	Model	Representation space	Encoder	Decoder
Transformation	TTransE(2018)	$\mathbb{R}^d$	-	$\|\boldsymbol{h}+\boldsymbol{r}+\boldsymbol{\tau}-\boldsymbol{t}\|_{1/2}$
	TA-TransE(2018)	$\mathbb{R}^d$	$r_{seq} = LSTM(r:\tau)$	$\|\boldsymbol{h}+\boldsymbol{r}_{seq}-\boldsymbol{t}\|_2$
	HyTE(2018)	$\mathbb{R}^d$	$\boldsymbol{h}_\tau = \boldsymbol{h} - (\boldsymbol{w}_\tau^T\boldsymbol{h})\boldsymbol{w}_\tau$	$\|\boldsymbol{h}_\tau+\boldsymbol{r}_\tau-\boldsymbol{t}_\tau\|_{1/2}$
	Tero(2020)	$\mathbb{C}^d$	$\boldsymbol{h}_\tau = \boldsymbol{h}\circ\boldsymbol{\tau}$	$\|\boldsymbol{h}_\tau+\boldsymbol{r}-\overline{\boldsymbol{t}}_\tau\|$
	ChronoR(2021)	$\mathbb{R}^{d\times k}$	$\boldsymbol{h}_{r,\tau} = \boldsymbol{h}\circ[\boldsymbol{r}\|\boldsymbol{\tau}]\circ\boldsymbol{r}_2$	$\langle\boldsymbol{h}_{r,\tau},\boldsymbol{t}\rangle$
	RotateQVS(2022)	$\mathbb{Q}^d$	$\boldsymbol{h}_\tau = \boldsymbol{\tau}\boldsymbol{h}\boldsymbol{\tau}^{-1}$	$\|\boldsymbol{h}_\tau+\boldsymbol{r}-\overline{\boldsymbol{t}}_\tau\|$
Decomposition	DE-SimplE(2020)	$\mathbb{R}^d$	$\boldsymbol{e}_\tau[n] = \begin{cases}\boldsymbol{e}[n]\sigma(\boldsymbol{w}[n]\tau+\boldsymbol{b}[n]), & 1\le n\le\gamma d\\ \boldsymbol{e}[n], & \gamma d\le n\le d\end{cases}$	$\frac{1}{2}(\langle\boldsymbol{h}_\tau,\boldsymbol{r},\boldsymbol{t}_\tau\rangle+\langle\boldsymbol{t}_\tau,\boldsymbol{r^{-1}},\boldsymbol{h}_\tau\rangle)$
	T-SimplE(2020)	$\mathbb{R}^d$	-	$\frac{1}{2}(\langle\boldsymbol{h},\boldsymbol{r},\boldsymbol{t},\boldsymbol{\tau}\rangle+\langle\boldsymbol{t},\boldsymbol{r^{-1}},\boldsymbol{h},\boldsymbol{\tau}\rangle)$
	TComplEx(2020)	$\mathbb{C}^d$	-	$re(\langle\boldsymbol{h},\boldsymbol{r},\overline{\boldsymbol{t}},\boldsymbol{\tau}\rangle)$
	TLT-KGE(2022)	$\mathbb{H}^d$	$\boldsymbol{h}_\tau^q = \boldsymbol{h_a}+\boldsymbol{h_b}\boldsymbol{i}+\boldsymbol{\tau}_{\boldsymbol{e},\boldsymbol{c}}\boldsymbol{j}+\boldsymbol{\tau}_{\boldsymbol{e},\boldsymbol{d}}\boldsymbol{k}$, $\boldsymbol{t}_\tau^q = \boldsymbol{t_a}+\boldsymbol{t_b}\boldsymbol{i}+\boldsymbol{\tau}_{\boldsymbol{e},\boldsymbol{c}}\boldsymbol{j}+\boldsymbol{\tau}_{\boldsymbol{e},\boldsymbol{d}}\boldsymbol{k}$, $\boldsymbol{r}_\tau^q = \boldsymbol{r_a}+\boldsymbol{r_b}\boldsymbol{i}+\boldsymbol{\tau}_{\boldsymbol{r},\boldsymbol{c}}\boldsymbol{j}+\boldsymbol{\tau}_{\boldsymbol{r},\boldsymbol{d}}\boldsymbol{k}$	$\langle\boldsymbol{h}_\tau^q\odot\boldsymbol{r}_\tau^q,\boldsymbol{t}_\tau^q\rangle+\langle(\boldsymbol{h}_\tau^q\odot\boldsymbol{r}_\tau^q)',\boldsymbol{t}_\tau^q\rangle$
	TuckERT(2022)	$\mathbb{R}^d$	-	$\langle\boldsymbol{\mathcal{M}};\boldsymbol{h},\boldsymbol{r},\boldsymbol{t},\boldsymbol{\tau}\rangle$
Graph Neural Networks	TEA-GNN(2021)	$\mathbb{R}^d$	*Time-aware GNN*	-
	TREA(2022)	$\mathbb{R}^d$	*Temporal Relational GAT*	-
	DEGAT(2022)	$\mathbb{R}^d$	*GAT*	*DE-ConvKB*
	L^2TKG(2023)	$\mathbb{R}^d$	*R-GCN + GRU*	*ConvTransE*
Capsule Network	TempCaps(2022)	$\mathbb{R}^d$	*Temporal Capsule Network*	*MLP*
	BiQCap(2023)	$\mathbb{H}^d$	*Biquaternion + Capsule Network*	*MLP*
	DuCape(2023)	$\mathbb{H}^d$	*Dual Quaternion + Capsule Network*	*MLP*
Autoregression	RE-NET(2020)	$\mathbb{R}^d$	*R-GCN + GRU*	*MLP*
	Glean(2020)	$\mathbb{R}^d$	*(CompGNC + GCN) + RNN*	-
	RE-GCN(2021)	$\mathbb{R}^d$	*relation-aware GCN + GRU*	*ConvTransE*
	TiRGN(2022)	$\mathbb{R}^d$	*relation-aware GCN + GRU*	*Time-ConvTransE*
	Cen(2022)	$\mathbb{R}^d$	*relation-aware GCN + GRU*	*CNN*
TPP	Know-Evolve(2017)	$\mathbb{R}^d$	*Rayleigh process*	-
	GHNN(2020)	$\mathbb{R}^d$	*Graph Hawkes Process*	-
	EvoKG(2022)	$\mathbb{R}^d$	*Structure module + Temporal module*	-
Interpretability	xERTE(2021)	$\mathbb{R}^d$	*TRGA*	-
	CluSTeR(2021)	$\mathbb{R}^d$	*RL + R-GCN + GRU*	-
	TiTer(2021)	$\mathbb{R}^d$	*RL*	-
Language model	ICLTKG(2023)	$\mathbb{R}^d$	-	*Transformer*
	zrLLM(2023)	$\mathbb{R}^d$	-	*Transformer*
	ECOLA(2022)	$\mathbb{R}^d$	-	*Transformer + Dyernie*
	GENTKG(2023)	$\mathbb{R}^d$	-	*Transformer*
	Chain of History(2024)	$\mathbb{R}^d$	-	*Transformer*
Few-shot Learning	MetaTKG(2022)	$\mathbb{R}^d$	*REGCN*	-
	MetaTKGR(2022)	$\mathbb{R}^d$	*Time-aware GAT*	-
	TR-Match(2023)	$\mathbb{R}^d$	*Time-Relation Attention*	-
	MTKGE(2023)	$\mathbb{R}^d$	*RPPG + TSPG + GCN*	-
Others	CygNet(2021)	$\mathbb{R}^d$	*Copy mode+ Generation mode*	-
	TANGO(2021)	$\mathbb{R}^d$	*MGCN + Trans + ODE Solver*	Distmult or TuckER
	Dyernie(2020)	$\mathbb{R}^d$	$\boldsymbol{e}_{\boldsymbol{\tau}}^{(\boldsymbol{i})} = exp_0^{K_i}(log_0^{K_i}(\overline{\boldsymbol{e}}^{(\boldsymbol{i})})+\boldsymbol{v}_{\boldsymbol{e}^{(\boldsymbol{i})}})$	$\sum_{i=1}^{k}-d_{M_{K_i}^{n_i}}(\boldsymbol{P}^{(\boldsymbol{i})}\otimes_{K_i}\boldsymbol{h}_{\boldsymbol{\tau}}^{(\boldsymbol{i})},$ $\boldsymbol{t}_{\boldsymbol{\tau}}^{(\boldsymbol{i})}\oplus_{K_i}\boldsymbol{p}^{(\boldsymbol{i})})+b_h^{(i)}+b_t^{(i)}$
	BoxTE(2022)	$\mathbb{R}^d$	$\boldsymbol{h}^{\boldsymbol{r}(\boldsymbol{h},\boldsymbol{t})} = \boldsymbol{h}+\boldsymbol{b}_{\boldsymbol{t}}$, $\boldsymbol{t}^{\boldsymbol{r}(\boldsymbol{h},\boldsymbol{t})} = \boldsymbol{t}+\boldsymbol{b}_{\boldsymbol{h}}$, $\boldsymbol{r}^{\boldsymbol{h}\|\boldsymbol{\tau}} = \boldsymbol{r}^{\boldsymbol{h}}-\boldsymbol{\tau}^{\boldsymbol{r}}$, $\boldsymbol{r}^{\boldsymbol{t}\|\boldsymbol{\tau}} = \boldsymbol{r}^{\boldsymbol{t}}-\boldsymbol{\tau}^{\boldsymbol{r}}$	$\|d(\boldsymbol{h}^{\boldsymbol{r}(\boldsymbol{h},\boldsymbol{t})},\boldsymbol{r}^{\boldsymbol{h}\|\boldsymbol{\tau}})\|+\|d(\boldsymbol{t}^{\boldsymbol{r}(\boldsymbol{h},\boldsymbol{t})},\boldsymbol{r}^{\boldsymbol{t}\|\boldsymbol{\tau}})\|$
	TGeomE(2023)	$\mathbb{G}^{n\times d}$	$\boldsymbol{h},\boldsymbol{r},\boldsymbol{t} = [M_1,...,M_k]$	$\langle Sc(\boldsymbol{h}\otimes_n\boldsymbol{r}\otimes_n\overline{\boldsymbol{t}}),1\rangle$

TKG datasets

Table 1: Statistics of Datasets for Temporal Knowledge Graphs in different applications

Applications	Datasets	$\|E\|$	$\|R\|$	$\|T\|$	$\|F\|$
Knowledge Reasoning	ICEWS18	23,033	256	304	468,558
	ICEWS14	7,128	230	365	90,730
	ICEWS05-15	10,488	251	4,017	461,329
	GDELT	7,691	240	2,751	2,278,405
	Wikidata	12,554	24	232	669,934
	YAGO	10,623	10	189	201,089
Entity Alignment	DICEWS	9,517/9,537	247/246	4,017	307,552/307,553
	YAGO-WIKI	49,626/49,222	11/30	245	221,050/317,814
Question Answering	ICEWS21	-	253	243	-
	Wikidata	432,715	814	1,726	7,224,361

TGB THG

- Continous time methods: HTGN-BTW and STHN.

HTGN-BTW , enabling TGN to accommodate heterogeneous node and edge types. STHN utilizes a link encoder and patching techniques to incorporate edge type and time information respectively.

- Discrete-time methods: random walk based methods and message-passing based methods.

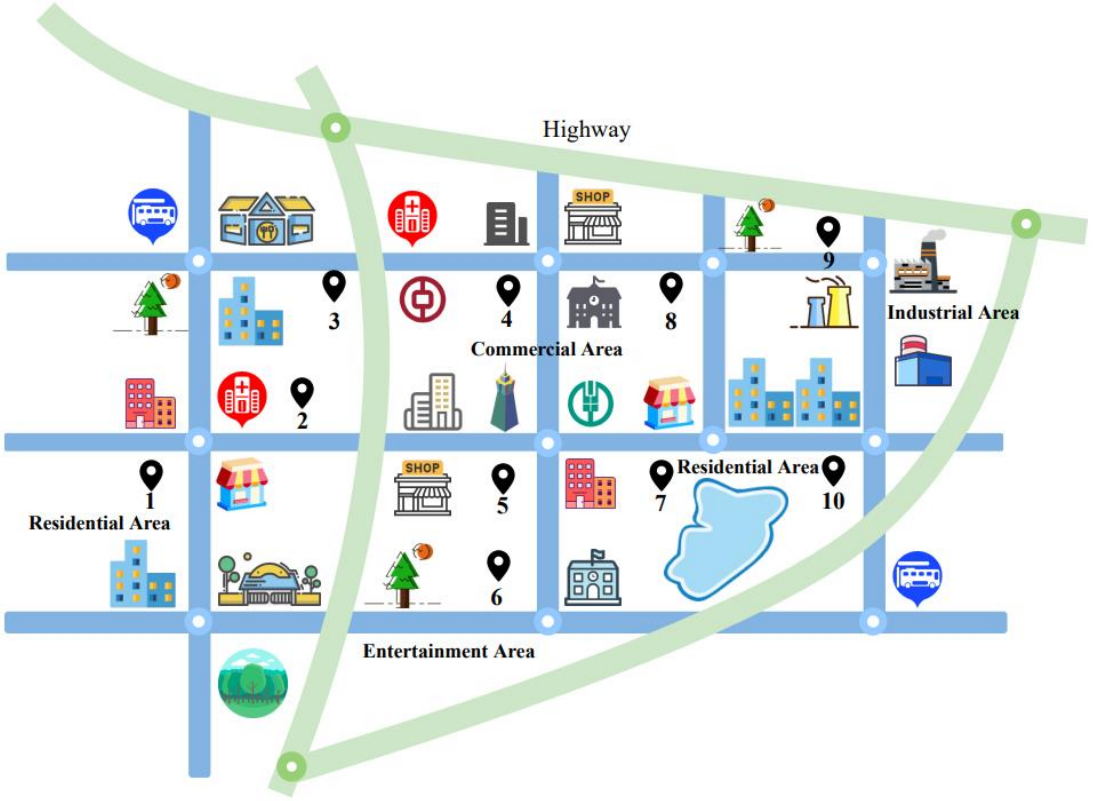
- Common THG datasets:

MathOverflow , **Netflix** and **Movielens** are small and only contain a few million edges.

Large datasets such as Dataset **A** and **B** from the WSDM 2022 Challenge and the **TRACE** and **THEIA** datasets.

large **thgl-myket** dataset with 53 million edges and 14 million timestamps.

Spatiotemporal Graphs and Graph Deep Learning for Time Series Processing



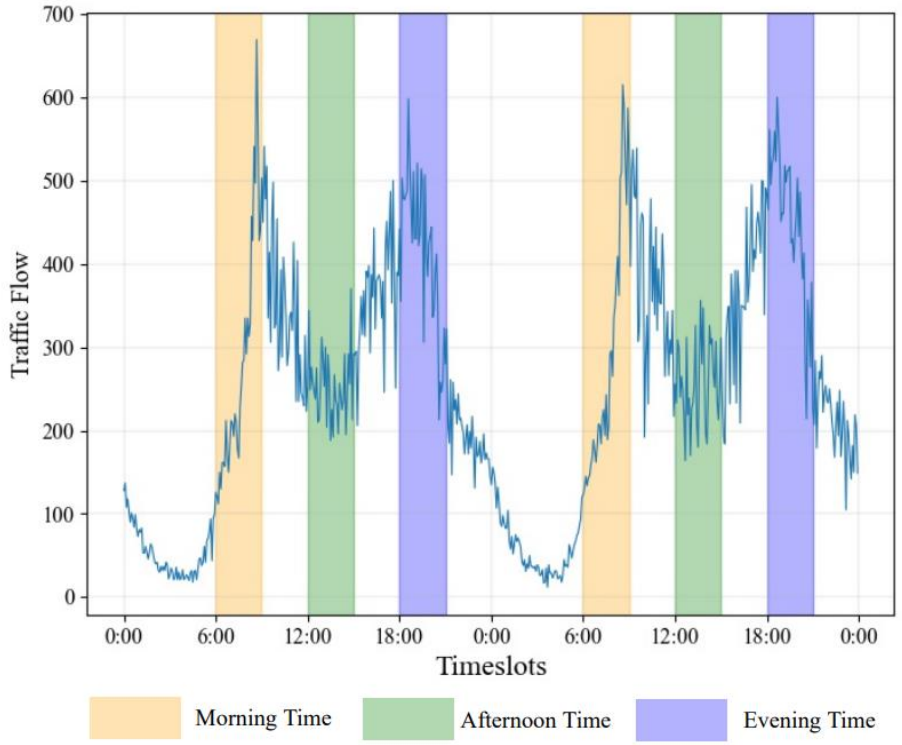


Table 2: Summary of Representative Models for Spatial and Temporal Modeling Techniques.

Venue	Model	Graph Construction	Spatial Components	Temporal Components
IJCAI 18	STGCN [19]	Distance-based Graph	ChebNet/GCN	Gated TCN
ICLR 18	DCRNN [21]	Distance-based Graph	DGC	GRU
AUAI 18	GaAN [55]	Distance-based Graph	GAT	GRU
IJCAI 19	GWNET [45]	Distance-based/Adaptive Graph	DGC	Dilated TCN
AAAI 19	ASTGCN [56]	Connectivity Graph	GCN; Attention	TCN; Attention
AAAI 19	ST-MGCN [7]	Multiple Graph	ChebNet	RNN
KDD 19	ST-MetaNet [57]	Distance-based graph	Meta-GAT	Meta-GRU
IJCAI 20	LSGCN [58]	Distance-based Graph	ChebNet; GAT	Gated TCN
AAAI 20	STSGCN [33]	Connectivity Graph	GCN	GCN
AAAI 20	GMAN [23]	Distance-based Graph	Embedding; Attention	Embedding; Attention
KDD 20	MTGNN [22]	Adaptive Graph	MHGC	Dilated TCN
NIPS 20	AGCRN [44]	Adaptive Graph	GCN	GRU
CIKM 20	STAG-GCN [35]	Multiple Graph	GCN; GAT	TCN; Self-Attention
TITS 20	T-MGCN [38]	Multiple Graph	GCN	GRU
AAAI 20	MRA-BGCN [53]	Distance-based/Edge Graph	MHGC	GRU
WWW 20	STGNN [17]	Distance-based Graph	GAT	GRU; Transformer
ICLR 21	GTS [47]	Sampled Graph	DGC	GRU
TKDD 21	DGCRN [8]	Dynamic Graph	MHGC	GRU
TKDE 21	ASTGNN [59]	Distance-based Graph	GAT	TCN; Transformer
AAAI 21	STFGNN [37]	Semantic Graph	GCN	GCN; Dilated TCN
AAAI 21	CCRNN [60]	Adaptive Graph	GCN	GRU
KDD 21	STGODE [36]	Distance-Based/Semantic Graph	GCN; ODE	TCN
KDD 21	DMSTGCN [61]	Dynamic Graph	DGC	Dilated TCN
IJCAI 22	RGSL [48]	Connectivity/Sampled Graph	GCN	GRU
KDD 22	ESG [62]	Dynamic Graph	GRU; DGC	Dilated TCN
KDD 22	STEP [63]	Sampled Graph	DGC	Transformer; Dilated TCN
AAAI 22	STG-NCDE [64]	Adaptive Graph	GCN; NCDE	NCDE
ICML 22	DSTAGNN [65]	Dynamic Graph	ChebNet; Attention	Gated TCN; Attention

***Note**: To simplify the presentation, we use the terms **"DGC"** to refer to Diffusion Graph Convolution and **"MHGC"** to refer to Multi-Hop Graph Convolution. We refer to a model with three or more graphs as a **"Multiple Graph"** model.

Libraries and Datasets

- 13 processed TG datasets accessible via "pip install dgb"
- Pytorch Geometric Temporal
- TGL library
- Chartalist Dynamic Blockchain Transaction Network
- Temporal knowledge graph forecasting benchmark
- TGB website and pypi install
- DyGLib Github
- Live Graph Lab website and dataset
- DistTGL Github
- Torch Spatiotemporal (TSL) website and Github
- pathpyG website and Github
- RelBench website and Github

- Pytorch Geometric Temporal
- TGL paper and Github
- DGB pypi install, paper and datasets
- Chartalist Blockchain Network website, paper and Github
- TKG Forecasting Evaluation paper and Github

Dataset	Type	Nodes	Edges	Granularity
Social Evolution	Social network	83	376-791 Associations 2,016,339 Communications	6 mins
Github	Social network	12,328	70,640-166,565 Associations 604,649 Communications	-
HEP-TH	Citation network	1,424-7,980	2,556-21,036	Monthly
Autonomous systems	Communication network	103-6,474[5]	243-13,233	Daily
GDELT	Events knowledge graph	14,018	31.29M	15 mins
ICEWS	Events knowledge graph	12498	0.67M	Daily
YAGO	Knowledge graph	15,403	138,056	Mostly yearly
Wikidata	Knowledge graph	11,134	150,079	Yearly
Reddit	Social network	55,863	858,49	Seconds
Enron	Email network	151	50.5K	Seconds
FB-Forum	Social network	899	33.7K	Seconds
Blog	Social network	5,196	171,743	-
Cora[6]	Citation network	2708	5429	-
Flicker	Social network	1,715,256	22,613,981	-
UCI	Communication network	1,899	59,835	Seconds
Radoslaw[7]	Email network	167	82.9K	Seconds
DBLP[89]	Citation network	315,159	743,70	-
YELP	Bipartite ratings	6,569	95,361	Seconds
MovieLens-10M	Bipartite ratings	20,537	43,760	Seconds
CONTACT	Face-to-face proximity	274	28,200	Seconds
HYPERTEXT09	Face-to-face proximity	113	20,800	Seconds
Elliptic[10]	Bitcoin transactions	203,769	234,355	49 time steps

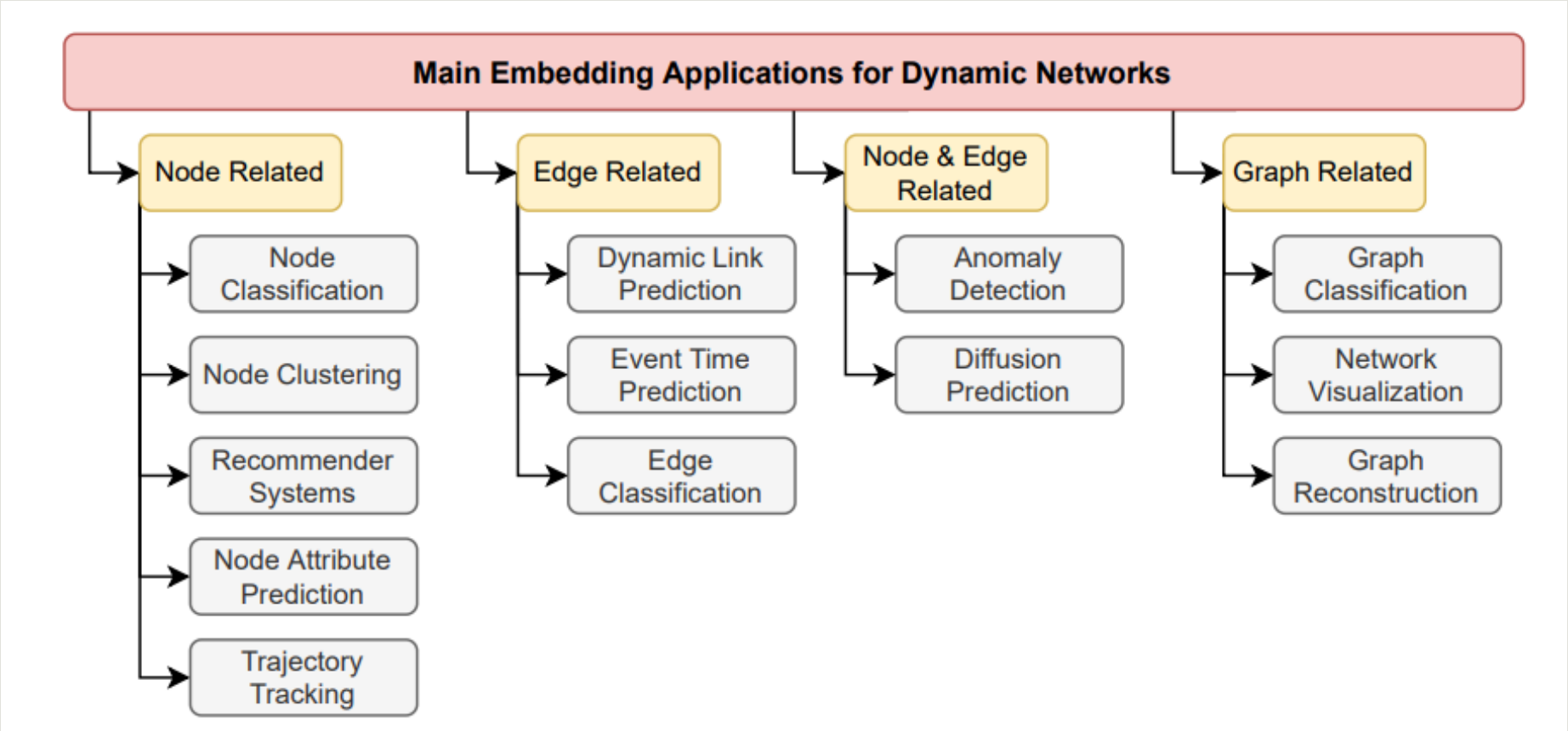
TGB dataset

Table 1: Dataset Statistics. Dataset names are colored based on their scale as small, medium, and large. ¶: Edges can be *Weighted*, *Directed*, or *Attributed*.

	Dataset	Domain	# Nodes	# Edges	# Steps	Surprise	Edge Properties¶
Link	tgbl-wiki	interact.	9,227	157,474	152,757	0.108	W: ✘, Di: ✓, A: ✓
	tgbl-review	rating	352,637	4,873,540	6,865	0.987	W: ✓, Di: ✓, A: ✘
	tgbl-coin	transact.	638,486	22,809,486	1,295,720	0.120	W: ✓, Di: ✓, A: ✘
	tgbl-comment	social	994,790	44,314,507	30,998,030	0.823	W: ✓, Di: ✓, A: ✓
	tgbl-flight	traffic	18143	67,169,570	1,385	0.024	W: ✘, Di: ✓, A: ✓
Node	tgbn-trade	trade	255	468,245	32	0.023	W: ✓, Di: ✓, A: ✘
	tgbn-genre	interact.	1,505	17,858,395	133,758	0.005	W: ✓, Di: ✓, A: ✘
	tgbn-reddit	social	11,766	27,174,118	21,889,537	0.013	W: ✓, Di: ✓, A: ✘
	tgbn-token	transact.	61,756	72,936,998	2,036,524	0.014	W: ✓, Di: ✓, A: ✓

surprise index (i.e., $\frac{|E_{test} \setminus E_{train}|}{|E_{test}|}$)

Applications

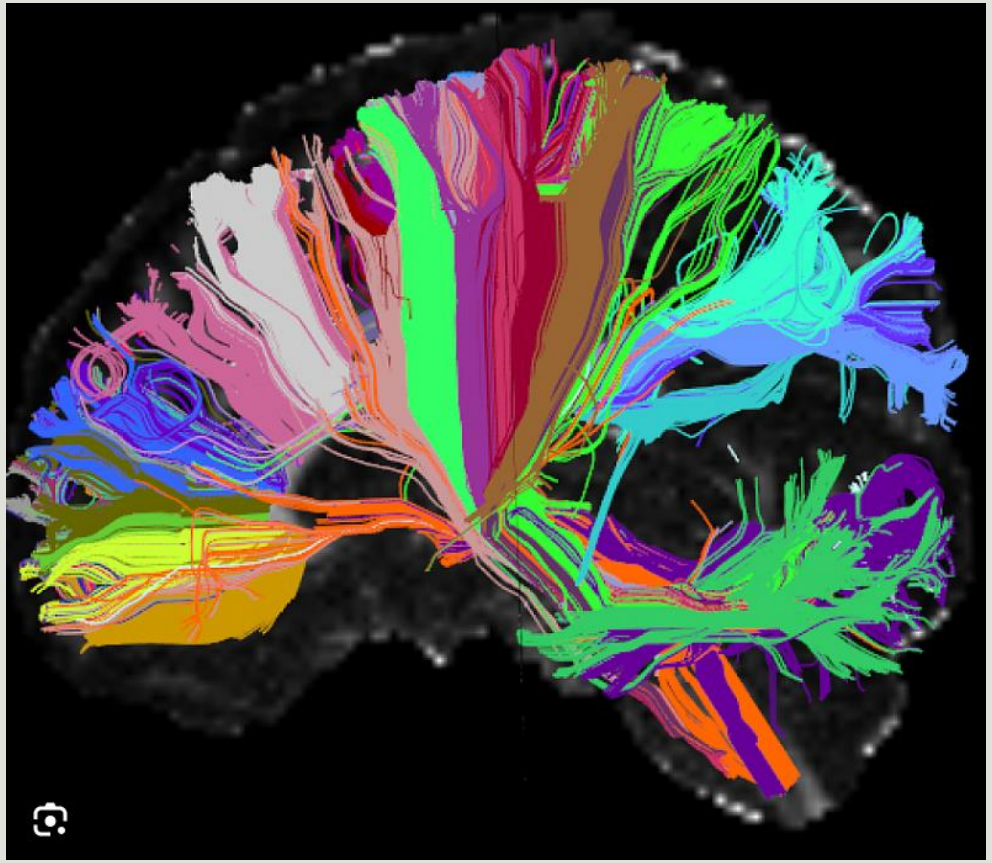


Biology and medicine

Brain connectome

Protein–protein interaction networks

Temporal graphs and single cell technologies

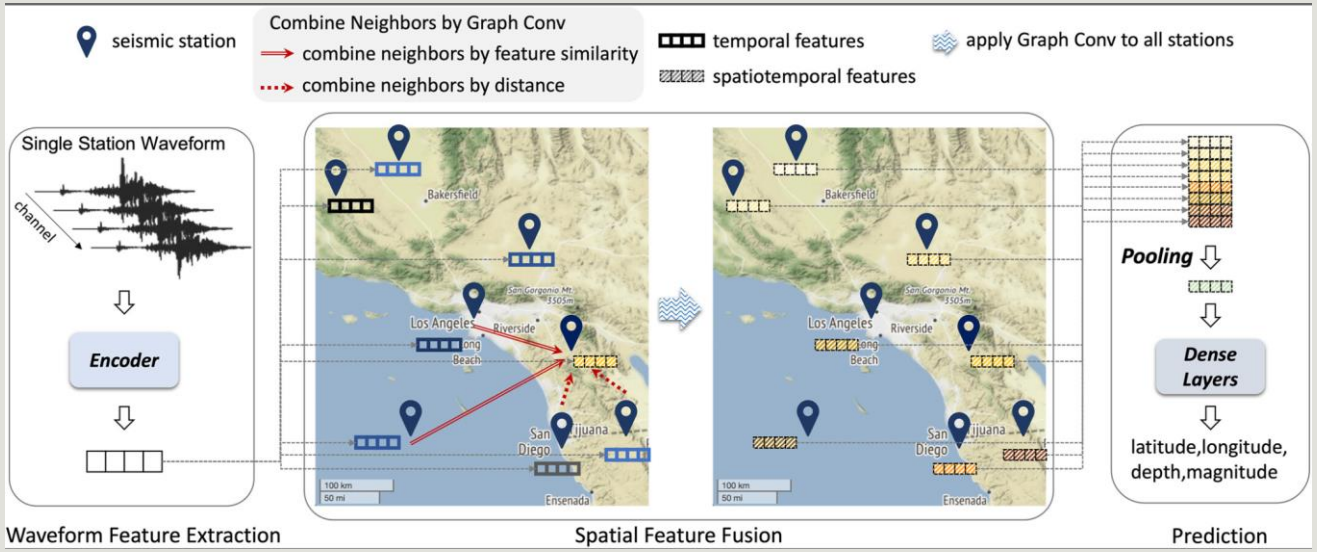


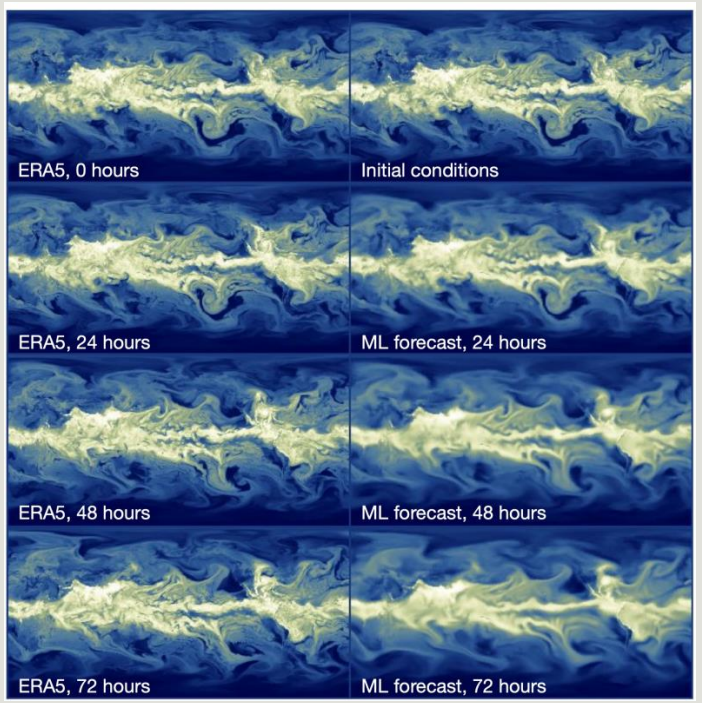
Weather forecasting, earthquake location and estimation

R. Keisler. Forecasting global weather with graph neural networks. arXiv preprint arXiv:2202.07575, 2022.

I.W. McBrearty and G.C. Beroza. Earthquake location and magnitude estimation with graph neural networks. In IEEE ICIP, 2022.

R. Keisler. Forecasting global weather with graph neural networks. arXiv preprint arXiv:2202.07575, 2022.



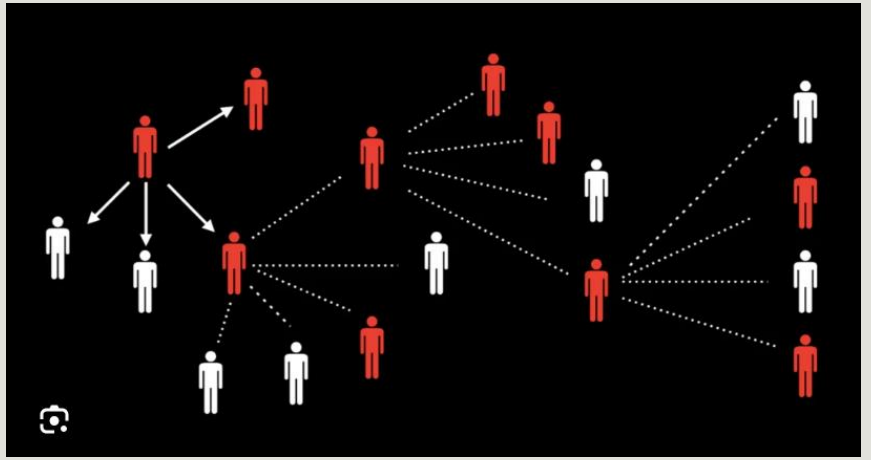


Epidemic modeling and contact tracing

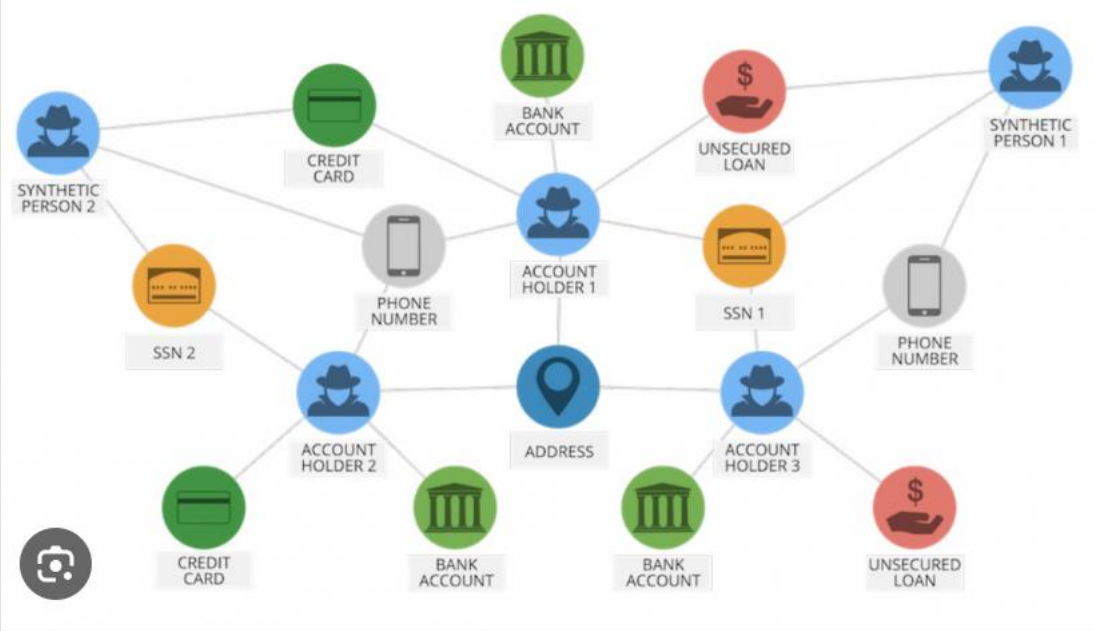
G. Cencetti, G. Santin, A. Longa, E. Pigani, A. Barrat, C. Cattuto, S. Lehmann, M. Salathe, and B. Lepri. Digital proximity tracing on empirical contact networks for pandemic control. Nature Communications, 2021.

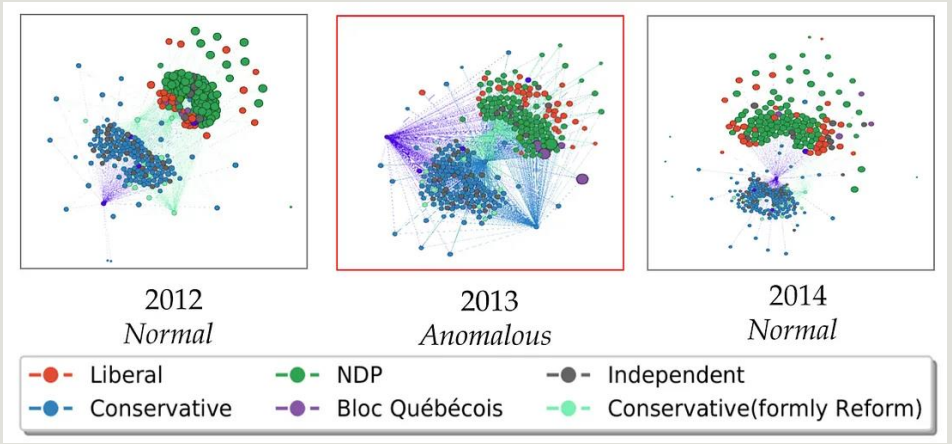
M. K.P. So, A. Tiwari, A. M.Y. Chu, J. T.Y. Tsang, and J. N.L. Chan. Visualizing covid19 pandemic risk through network connectedness. International Journal of Infectious Diseases, 96:558–561, Jul 2020.

A. Micheli and D. Tortorella. Discrete-time dynamic graph echo state networks. Neurocomputing, 496:85–95, 2022.



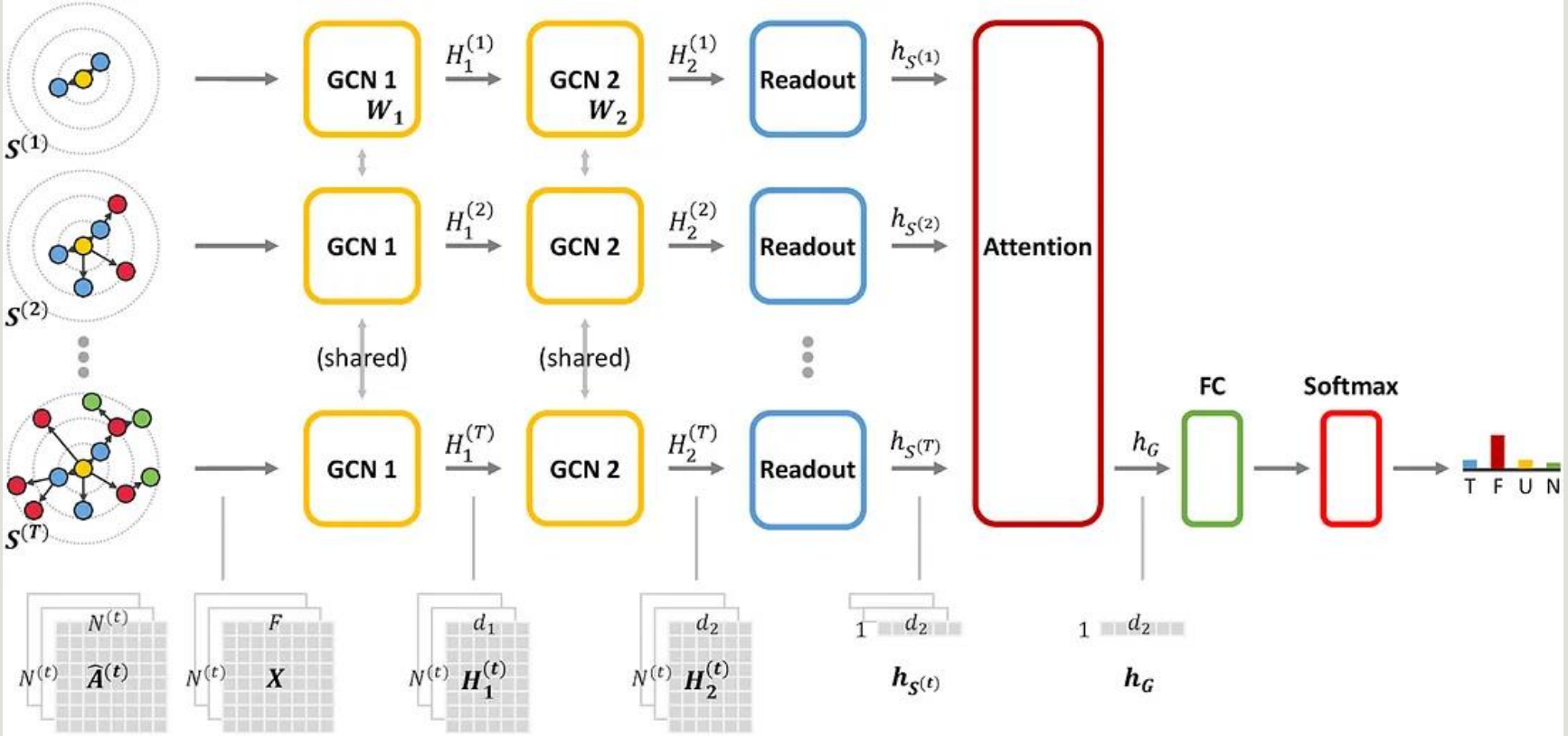
Anomaly Detection



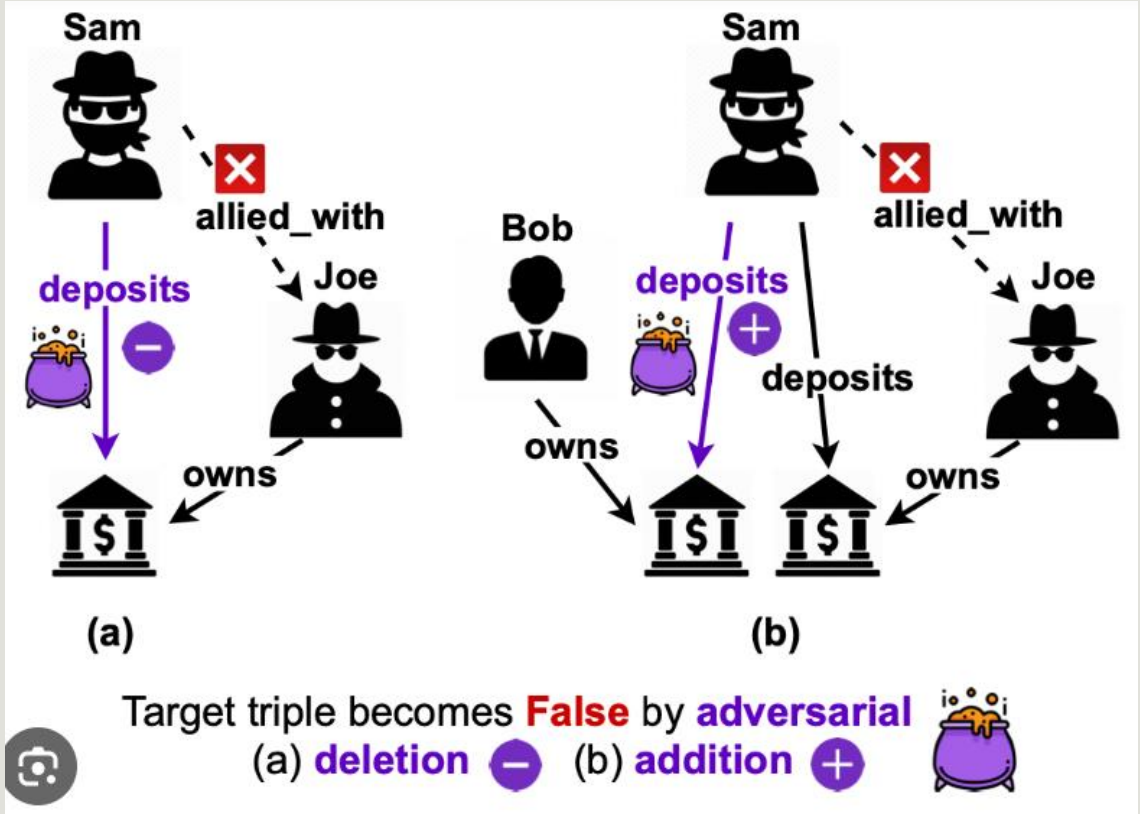


Detecting Misinformation

DynGCN



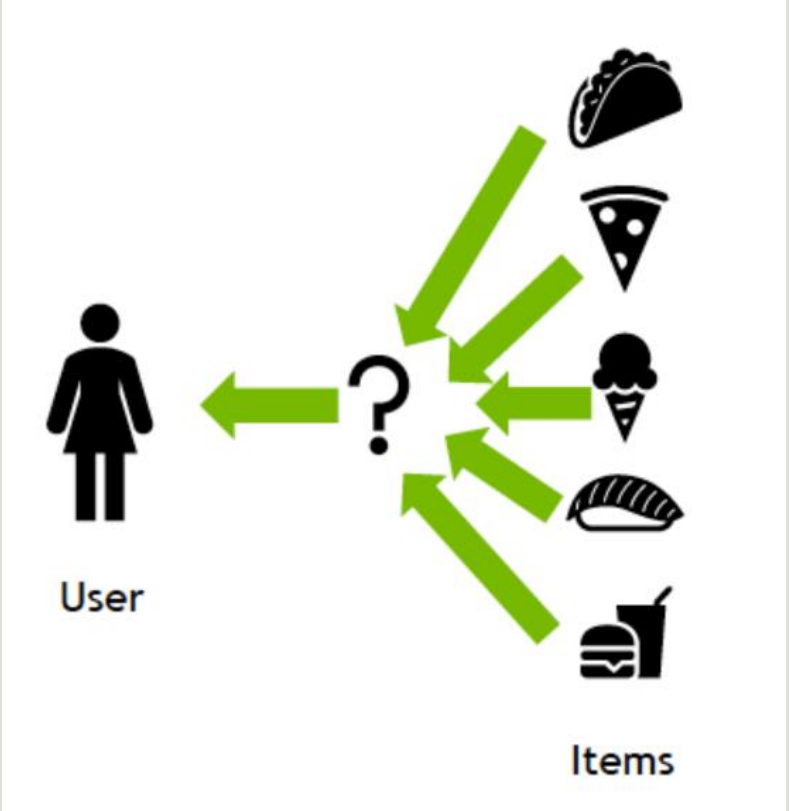
Adversarial Attacks



Recommendation systems

C. Gao, X. Wang, X. He, and Y. Li. Graph neural networks for recommender system. WSDM ’22, page 1623–1625, New York, NY, USA, 2022. Association for Computing Machinery.

S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui. Graph neural networks in recommender systems: a survey. ACM CSUR, 2022.



Social network analysis

S. Deng, H. Rangwala, and Y. Ning. Learning dynamic context graphs for predicting social events. In ACM SIGKDD, 2019.

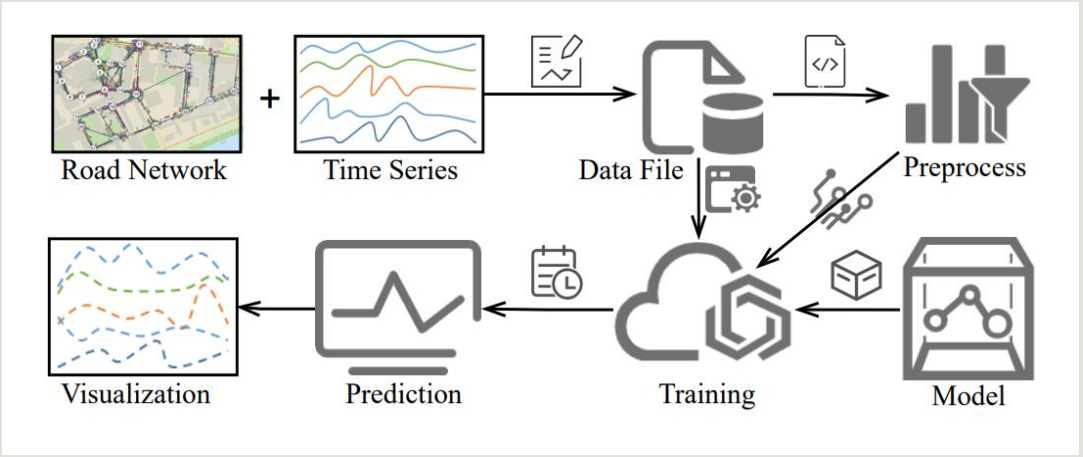
W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin. Graph neural networks for social recommendation. WWW '19, New York, NY, USA, 2019. Association for Computing Machinery.

Transportation systems

Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. Expert Systems with Applications, 207:117921, 2022.

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

A. Cini, I. Marisca, F.M. Bianchi, and C. Alippi. Scalable spatiotemporal graph neural networks. arXiv preprint arXiv:2209.06520, 2022.



More references

W. Cong, S. Zhang, J. Kang, B. Yuan, H. Wu, X. Zhou, H. Tong, and M. Mahdavi. Do we really need complicated model architectures for temporal networks? arXiv preprint arXiv:2302.11636, 2023.

2020. Representation Learning for Dynamic Graphs A Survey

2021. A Survey on Embedding Dynamic Graphs

2021. Foundations and Modeling of Dynamic Networks Using Dynamic Graph Neural Networks A Survey

2022. A Survey on Graph Representation Learning Methods

2022. A Survey on Temporal Graph Representation Learning and Generative Modeling

2023. Graph Neural Networks for temporal graphs State of the art open challenges and opportunities

2023. Temporal Graph Benchmark for Machine Learning on Temporal Graphs

2024 TGB 2 A Benchmark for Learning on Temporal Knowledge Graphs and Heterogeneous Graphs

2024. A Survey on Temporal Knowledge Graph Representation Learning and Applications

Thank you

Spectral GCN

Temporal point process

A temporal point process is a point process that can be represented as a counting process N(t), recording the number of events up to time t, thus being useful for modeling sequential asynchronous discrete events occurring in continuous time.

Temporal graph representation learning has leveraged the use of temporal point processes as well. Temporal point processes are stochastic processes employed for modeling sequential asynchronous discrete events occurring in continuous time in “Multivariate point processes” paper.

Model type	Model name	Encoder	Link addition	Link deletion	Node addition	Node deletion	Network type
Discrete networks							
Stacked DGNN	GCRN-M1 [69]	Spectral GCN [74] & LSTM	Yes	Yes	No	No	Any
	WD-GCN [76]	Spectral GCN [75] & LSTM	Yes	Yes	No	No	Any
	CD-GCN [76]	Spectral GCN [75] & LSTM	Yes	Yes	No	No	Any
	RgCNN [83]	Spatial GCN [84] & LSTM	Yes	Yes	No	No	Any
	DyGGNN [85]	GGNN [86] & LSTM	Yes	Yes	No	No	Any
	DySAT [17]	GAT [88] & temporal attention [89]	Yes	Yes	Yes	Yes	Any
	TNDCN [25], [90]	Spectral GCN [25] & TCN [25]	Yes	Yes	No	No	Any
	StrGNN [93]	Spectral GCN [75] & GRU	Yes	Yes	No	No	Any
	HDGNN [24]	Spectral GCN [75] & A variety of RNNs	Yes	Yes	Yes	Yes	Heterogeneous
	TeMP [22]	R-GCN [97] stacked with either GRU or attention	Yes	Yes	No	No	Knowledge
Integrated DGNN	GCRN-M2 [69]	GCN [74] integrated in an LSTM	Yes	Yes	No	No	Any
	GC-LSTM [20]	GCN [74] integrated in an LSTM	Yes	Yes	No	No	Any
	EvolveGCN [16]	LSTM integrated in a GCN [75]	Yes	Yes	Yes	Yes	Any
	LRGCN [23]	R-GCN [97] integrated in an LSTM	Yes	Yes	No	No	Any
	RE-Net [95]	R-GCN [97] integrated in several RNNs	Yes	Yes	No	No	Knowledge
	TNA [96]	GCN [75] stacked with a GRU and a linear layer	Yes	Yes	No	No	Any
Continuous networks							
RNN based							
	Streaming GNN [78]	Node embeddings maintained by architecture consisting of T-LSTM [108]	Yes	No	Yes	No	Directed strictly evolving
	JODIE [77]	Node embeddings maintained by an RNN based architecture	Yes	No	No	No	Bipartite, interaction
TTP based							
	Know-Evolve [21]	TPP parameterised by an RNN	Yes	No	No	No	Interaction, knowledge network
	DyREP [48]	TPP parameterised by an RNN aided by structural attention	Yes	No	Yes	No	Interaction combined with strictly evolving
	LDG [109]	TPP, RNN and self-attention	Yes	No	Yes	No	Interaction combined with strictly evolving
	GHN [111]	TPP parameterised by a continuous time LSTM [112]	Yes	No	No	No	Interaction, knowledge network
Time embedding based							
	TGAT [18]	Temporal [113] and structural [88] attention	Yes	No	Yes	No	Directed or undirected interaction
	TGN [19]	Temporal [113] and structural [88] attention with memory	Yes	No	Yes	No	Directed or undirected interaction

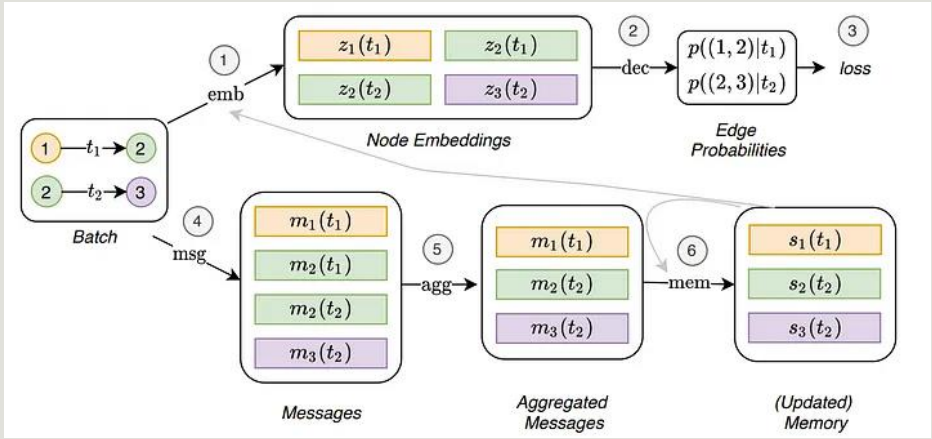
TGNs

- Generalize Message Passing Neural Networks (MPNNs) to temporal graphs.
- Generalizes Joint Dynamic User-Item Embeddings (JODIE) and Temporal Graph Attention (TGAT)

- Node memory which represents the state of the node at a given time, acting as a compressed representation of the node's past interactions.

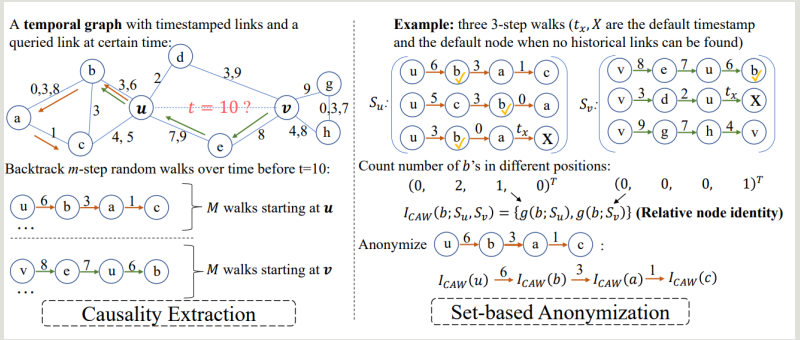
- Every time two nodes are involved in an interaction, they send messages to each other which are then used to update their memories.

- When computing the embedding of a node, an additional graph aggregation is performed over the temporal neighbors of the node, using both the original node features and memory at that point in time.



Walk Aggregating methods

Causal Anonymous Walks (CAW) rely on (temporal) random walks. In particular, to predict the existence of a link (u, v) at time t, CAW first extracts multiple random walks starting from u and v such that the timestamps of edges in a walk can only be monotonically decreasing. The walks are first anonymized by replacing each node identifier with a count vector of how many times that node appears at each of the possible positions in the walks. Each walk is then encoded using an RNN, and the encodings are then aggregated by using self-attention or taking a simple average.

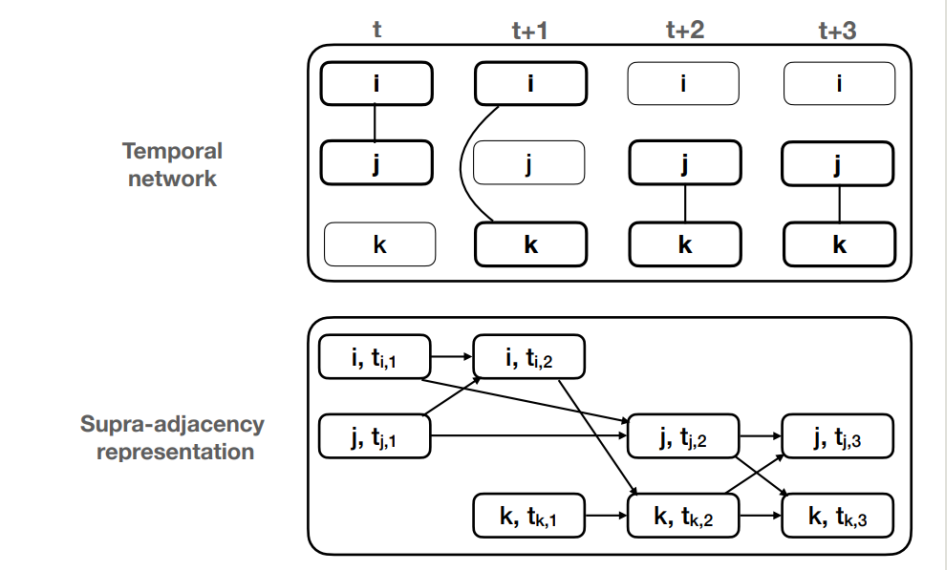


temporal random walks are exploited to efficently and automatically sample temporal network motifs, i.e, connected subgraphs with links that appear within a restricted time range.

Similarly, in “Towards fine-grained temporal network representation via time-reinforced random walk” a time-reinforced random walk is proposed to effectively sample the structural and temporal contexts over graph evolution.

DyANE

the temporal graphs are transformed into a static graph representation called a supra-adjacency representation. In this approach, the nodes are defined as (node, time) pairs from the original temporal graph. This static graph representation retains the temporal paths of the original network, crucial for comprehending and constraining the underlying dynamical processes. Afterwards, standard embedding techniques for static graphs, utilizing random walks, are employed.



Other methods

In “Deep temporal reasoning for dynamic knowledge graphs“ instead, the occurrence of an edge in a temporal graph is modeled as a multivariate point process, where the intensity function is influenced by the score assigned to that edge, which is computed using the learned entity embeddings. The entity embeddings, which evolve over time, are acquired through a recurrent architecture.

Also non-negative matrix factorization (NMF) has been employed for the purpose of link prediction in temporal graphs. In “Link prediction in dynamic networks based on non-negative matrix factorization”, novel iterative rules for NMF are proposed to construct the matrix factors that capture crucial features of the temporal graph, enhancing the accuracy of the link prediction process.

In STAR, a spatio-temporal attentive recurrent network model, is proposed for interpretable temporal node classification.

Other methods

In “Spatiotemporal deep graph infomax.“ a node-level regression task is achieved by training embeddings to maximize the mutual information between patches of the graph, at any given time step, and between features of the central nodes of patches, in the future.

Finally, TSNet is a comprehensive framework for node classification in temporal graphs, consisting of two key steps. Firstly, the graph snapshots undergo a sparsification process using edge sampling, guided by a learned distribution derived from the supervised classification task. This step effectively reduces the density of the snapshots. Subsequently, the sparsified snapshots are aggregated and processed through a convolutional network to extract meaningful features for node classification.

Lastly, Suresh et al. pointed out that existing methods maximizes accuracy independently over future links, ignoring the fact that future links often have dependency between each other. This is seen when a user selects among a list of items to purchase or a set of users to connect in a social network. Therefore, Suresh et al. treat dynamic link prediction as a ranking problem and propose Temporal Graph network for RANKing (TGRank) to learn to rank over a list of candidates. The task query now contains a center node s (in the example) with a set of candidate nodes (all other nodes in the subgraph) and the goal is to rank the most likely candidate as the destination of node s. To this end, TGRank follows three steps. First, the node s is labeled differently from other nodes. Then, GNNs are used to diffuse the center node label to every ranking candidate. This parametrized label diffusion step aggregates timestamps, multiplicity as well as features of historical interactions along the network from the center node to all candidates and provides provably more expressive power for link prediction. Lastly, a list-wise loss is used to optimize the ranking amongst candidates jointly. Empirically, it is also shown that with a listwise ranking loss, popular models such as TGN and TGAT also perform better than their original setup with binary classification loss.

Node-based and edge-based

Node-based models such as TGN [35], DyRep [42] and TCL [45] first leverage the node information such as temporal neighborhood or previous node history to generate node embeddings and then aggregate node embeddings from both source and destination node of an edge to predict its existence.

In comparison, edge-based methods such as CAWN [46] and GraphMixer [9] aim to directly generate embeddings for the edge of interest and then predict its existence. Lastly, the simple memory-based heuristic EdgeBank [33] without any learning component has shown surprising performance based on existing evaluation

Lastly, the simple memory-based heuristic EdgeBank [33] without any learning component has shown surprising performance based on existing evaluation

6.1.1 Model Evolution methods We call Model Evolution the evolution of the parameters of a static GNN model over time. This mechanism is appropriate for modelling STG, as the evolution of the model is performed at the snapshot level. More formally, these methods learn an embedding hv(ti) = GNN(v, Gi ; θ(ti)), where θ(ti) = REC(θ(ti−j) : 1 ≤ j ≤ imax) is a parameter-evolution network, and imax is the memory length. To the best of our knowledge, the only existing method belonging to this category is EvolveGCN [61]. This model utilizes a Recurrent Neural Network (RNN) to update the Graph Convolutional Network (GCN) [39] parameters at each time-step, allowing for model adaptation that is not constrained by the presence or absence of nodes. The method can effectively handle new nodes without prior historical information. A key advantage of this approach is that the GCN parameters are no longer trained directly, but rather they are computed from the trained RNN, resulting in a more manageable model size that does not increase with the number of time-steps. The paper presents two versions of this method: EvolveGCN-O uses a Long ShortTerm Memory (LSTM) to simply evolve the weights in time, while EvolveGCN-H represents the weights as hidden states of a Gated Recurrent Unit (GRU), whose input is the previous node embedding. 10 6.1.2 Embedding Evolution methods Rather than evolving the parameters of a static GNN model, Embedding Evolution methods focus on evolving the embeddings produced by a static model. This means to learn a node embedding hv(ti) = REC(hv(ti−j), i = 1, . . . , tmax) as the evolution of previous embeddings, where hv(ti−j) = GNN(v, Gi−j ; θ) are GNN embeddings for the SG Gi−j . There are several different TGNN models that fall under this category. These networks differ from one another in the techniques used for processing both the structural information and the temporal dynamics of the STGs. DySAT [70] introduces a generalization of Graph Attention Network (GAT) [81] for STGs. First, it uses a self-attention mechanism to generate static node embeddings at each timestamp. Then, it uses a second self-attention block to process past temporal embeddings for a node to generate its novel embedding. Decoupling graph evolution into two modular blocks allows for efficient computations of temporal node representations. The structural and temporal self-attention layers, combined and stacked, enable flexibility and scalability. The VGRNN model [29] uses VGAE [40] on each snapshot, where the latent representations are conditioned on a state variable modelled by Semi-Implicit Variational Inference (SIVI) [98] to handle the variation of the graph over time. The learned latent representation is then evolved through an LSTM conditioned on the previous time's latent representation, allowing the model to predict the future evolution of the graph. ROLAND [100] is a general framework for extending state-of-the-art GNN techniques to STGs. The key insight is that node embeddings at different GNN layers can be viewed as hierarchical node states. To generalize a static GNN for dynamic settings, hierarchical node states are updated based on newly observed nodes and edges through a Gated Recurrent Unit (GRU) update module [9]. The paper presents two versions of the model: ROLANDMLP, which uses a 2-layer MLP to update node embeddings, and ROLAND-moving average, which updates the node embeddings through the moving average among previous node embeddings. Finally, reservoir computing techniques have also been proposed. DynGESN [55] presents a method where each node embedding is updated by a recurrent mechanism using its temporal neighborhood and previous embedding, with fixed and randomly initialized recurrent weights. SSGNN [10] follows a similar approach but introduces trainable parameters in the decoder and combines randomized components in the encoder: initially, the encoder creates representations of the time series data observed at each node, by utilizing a reservoir that captures dynamics at various time scales; these representations are then further processed to incorporate spatial dynamics dictated by the graph structure. 6.2 Event-based models Models belonging to the Event-based macro category are designed to process ETGs (see Def. 5). These models are able to process streams of events by incorporating techniques that update the representation of a node whenever an event involving that node occurs, and they are an extension of message passing to TGs, since they combine and aggregate node representations over temporal neighborhoods. The models that lie in this macro category can be further classified in Temporal Embedding and Temporal Neighborhood methods, based on the technology used to learn the time dependencies. In particular, the Temporal Embedding models use recurrent or selfattention mechanisms to model sequential information from streams of events, while also incorporating a time encoding. This allows for temporal signals to be modeled by the interaction between time embedding, node features and the topology of the graph. Temporal Neighborhood models, instead, use a module that stores functions of events involving a specific node at a given time. These values are then aggregated and used to update the node representation as time progresses. 6.2.1 Temporal Embedding methods Temporal embedding methods model TGs by combining time embedding, node features, and graph topology. These models use an explicit functional time encoding, i.e., a vector embedding gt of time 11 based on Random Fourier Features (RFF) [64], which is translation-invariant (i.e., it depends only on the elapsed and not the absolute time). They extend the message passing architecture to temporal neighborhoods, where the time is encoded by gt, i.e., hv(t) = COMBINE((hv(t), g0), AGGREGATE({(hu(t ′), gt−t ′), u ∈ NT [v]})) where t ′ is the time of the connection event between u and v. Thus, gt−t ′ encodes the time elapsed between the current time t and the time of connection between u and v. TGAT [92], for example, introduces a graphtemporal attention mechanism which works on the embeddings of the temporal neighbors of a node, where the positional encoding is replaced by a temporal encoding based on RFFs. In addition, [92] implement a version of TGAT with all temporal attention weights set to an equal value (Const-TGAT). On the other hand, NAT [50] collects the temporal neighbors of each node into dictionaries, and then it learns the node representation with a recurrent mechanism, using the historical neighborhood of the current node and a RFF based time embedding. Note that [50] propose a dedicated data structure to support parallel access and update of the dictionary on GPUs. 6.2.2 Temporal Neighborhood methods The Temporal Neighborhood class includes all TGNN models that make use of a special mailbox module to update node embeddings based on events. When an event ε occurs, a function is evaluated on the details of the event to compute a mail or a message mε. For example, when a new edge appears between two nodes, a message is produced, taking into account the time of occurrence of the event, the node features, and the features of the new edge. The node representation is then updated at each time by aggregating all the generated messages. In more details, these methods extend message passing by learning an embedding hv(t) = COMBINE(hv(t), AGGREGATE(mε, ε = ε ± E = (u, t′) with u ∈ NT [v])), where ε ± E , with u ∈ NT [v], is the addition or deletion of a temporal neighbor of v. Several existing TGNN methods belong to this category. APAN [83] introduces the concept of asynchronous algorithm, which decouples graph query and model inference. An attention-based encoder maps the content of the mailbox to a latent representation of each node, which is decoded by an MLP adapted to the downstream task. After each node update following an event, mails containing the current node embedding are sent to the mailboxes of its neighbors using a propagator. DGNN [51] combines an interact module — which generates an encoding of each event based on the current embedding of the interacting nodes and its history of past interactions — and a propagate module — which transmits the updated encoding to each neighbors of the interacting nodes. The aggregation of the current node encoding with those of its temporal neighbors uses a modified LSTM, which permits to work on non-constant timesteps, and implements a discount factor to downweight the importance of remote interactions. TGN [66] provides a generic framework for representation learning in ETGs, and it makes an effort to integrate the concepts put forward in earlier techniques. This inductive framework is made up of separate and interchangeable modules. Each node the model has seen so far is characterized by a memory vector, which is a compressed representation of all its past interactions. Given a new event, a mailbox module computes a mail for every node involved. Mails will then be used to update the memory vector. To overcome the so-called staleness problem [37], an embedding module computes, at each timestamp, the node embeddings using their neighborhood and their memory states. Finally, TGL [106] is a general framework for training TGNNs on graphs with billions of nodes and edges by using a distributed training approach. In TGL, a mailbox module is used to store a limited number of the most recent interactions, called mails. When a new event occurs, the node memory of the relevant nodes is updated using the cached messages in the mailbox. The mailbox is then updated after the node embeddings are calculated. This process is also used during inference to ensure consistency in the node memory, even though updating the memory 12 is not required during this phase.

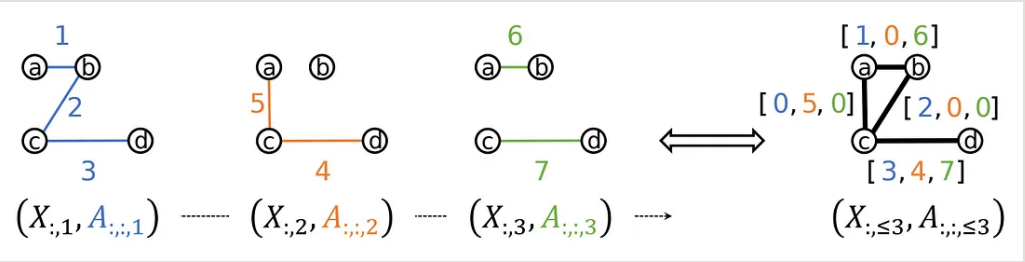
Category comparison

The categories of models identified in our taxonomy exhibit various strengths, weaknesses, or suitability for specific scenarios. First and foremost, the comparison between the macro categories of Snapshotbased and Event-based methods is straightforward, hinging on the choice of temporal graph representation, namely STGs or ETGs. Within each macro category, sub-categories exhibit their own set of advantages and disadvantages. For instance, in the Model Evolution category, learning the evolution of GNN parameters becomes complex when the GNN has a large number of parameters. On the other hand, the Embedding Evolution category has a limitation in that temporal learning exclusively relies on recurrent mechanisms, which may not fully guarantee the preservation of temporal correlations among substructures. Despite this drawback, the approach offers simplicity and intuitiveness, allowing for the exploration and evaluation of various recurrent mechanisms. In Temporal Embedding methods, defining the time encoding function is not trivial because it should capture different aspects of the graph, such as the temporal periodicity of interactions, recurrent interactions over time, and more. One advantage of this category, though, is that ad hoc time encoding functions can be defined, depending on the application domain. Finally, for the Temporal Neighborhood category, constructing the mailbox can be complex. For example, dense graphs have large mailboxes, which necessitate managing scalability issues. A specific mechanism to decide which nodes to include in the mailbox needs to be carefully designed, and this mechanism may also be domain-dependent. Similarly to the Temporal Embedding category, a benefit of the Temporal Neighborhood models is their ability to define domain-specific mailbox mechanisms. In summary, each category of models for temporal graph learning has its own set of advantages and disadvantages. Choosing a category depends on the representation of the input graph, the complexity of learning the temporal dynamics, the need for domainspecific encoding or mailbox mechanisms, and scalability considerations.

Expressiveness of Temporal Graph Networks

Xu et al. 2019 first characterized the discriminative power of Graph Neural Networks (GNNs) by connecting them to the Weisfeiler-Lehman (WL) graph isomorphism test and showing that many GNNs are no more powerful than the 1-WL test. Subsequent more expressive models such as subgraph GNNs, graph transformers and higher order GNNs are designed to be more expressive than 1-WL test.

Ribeiro et al. where the key idea is to categorize existing TGL methods into time-and-graph and time-then-graph frameworks.



1). In time-and-graph, GNNs are used to generate node embeddings on the snapshot graph at each time thus forming a sequence of node embeddings.

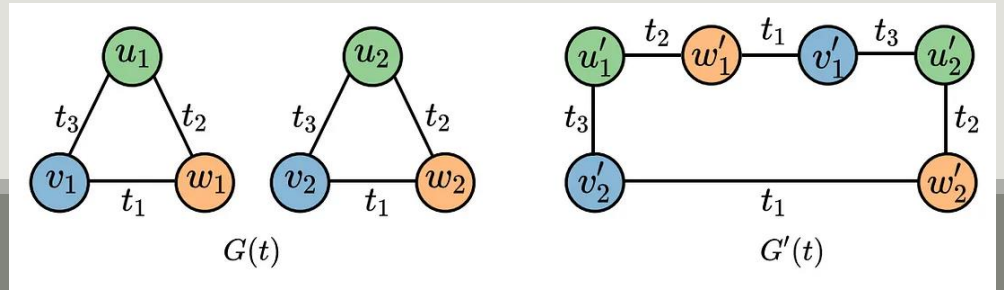
2). In time-then-graph, each edge in the TG is converted to a time series which indicates at which time the edge exists, therefore collapsing the temporal edges into edge features in a static graph.

Expressiveness of Temporal Graph Networks

It was shown that a time-then-graph representation can be constructed from any given time-and-graph representation thus proving time-then-graph is at least as expressive as time-and-graph. With the static representation in time-then-graph, we can directly use the WL-test expressiveness framework from the static graph for TGL methods. In this way, time-then-graph is more expressive than time-and-graph as long as a 1-WL GNN is used as the backbone model.

Souza et al. also aims to establish the 1-WL expressiveness framework for TGL methods. Notably, they view a CTDG as a sequence of time-stamped multi-graphs where the multi-graph G(t) at a given time t is obtained by sequentially applying all events prior to t. A multi-graph here means there can be multiple edges between two nodes in the graph and the edge attribute is the timestamp information.

Now, the Temporal WL test can be defined by applying the WL test on the multigraphs constructed from CTDGs. Therefore, more expressive TGN methods must be injective on its temporal neighborhood (i.e. hashing two different multi-set nodes into different colors), called injective MP-TGNs. Souza et al. also analyzed walk based TGNs such as CAW and show that MP-TGNs and CAW are not more expressive than each other (as seen above). Their proposed PINT method combines benefits from both categories of methods thus being the most expressive. The example below shows two temporal graphs that MP-TGNs are unable to distinguish. The colors are node labels and the edge has timestamps starting from t_1.



Explainable Temporal Graph Methods

Xia et al. proposed T-GNNExplainer as the first explainer designed for temporal graph models. T-GNNExplainer is model-agnostic and finds important events from a set of candidate events to best explain the model prediction. Xia et al. treat the problem of identifying a subset of explaining events as a combinatorial optimization problem by searching over a subset of the temporal graph within a given size. To tackle this, T-GNNExplainer employs an explorer-navigator framework. The navigator is trained from multiple target events to capture inductive correlations between events while the explorer searches out a specific combination of events based on Monte Carlo Tree Search, including node selection, node expansion, reward simulation and backprop. Which events are pruned is inferred from the navigator. The diagram below shows the framework of T-GNNExplainer.

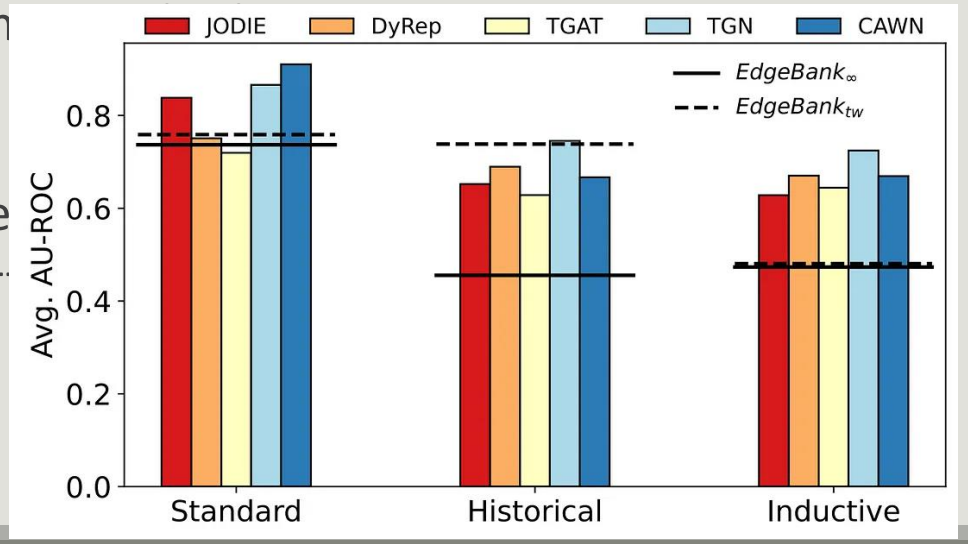
Recently, Chen et al. argued that to form human intelligible explanations for temporal graph events requires the explanation to be events that are temporally proximate and spatially adjacent to that of the prediction, referred to as cohesive explanations. Utilizing temporal motifs, recurring substructures within the temporal graph, is a natural solution to form cohesive explanations for temporal graphs. This is because temporal motifs are crucial factors that guide the generative process of future events. In the following example, the preferential attachment rule (often facilitating influencer effect in e-commerce) and triadic closure rule (explains common-friend rules in social networks) forms cohesive and plausible explanations.

Therefore, Chen et al. proposed TempME, a novel Temporal Motif-based Explainer to identify important temporal motifs to explain temporal GNNs. The framework of TempME is shown in the below figure. First, temporal motifs surrounding the target prediction are extracted. Then these candidate motifs are encoded via the Motif Encoder which leverages event anonymization, message passing and graph pooling to generate an embedding for each motif. Then, based on the Information-bottleneck principle, TempME assigns importance scores to these motifs constrained by both explanation accuracy and information compression. Lastly, explanations are constructed by sampling from the Bernoulli distribution based on the importance score.

Rethinking Evaluation in Temporal Graphs

Now, what can be considered as hard negative edges? First, we introduce the historical negative edges which are edges that appeared in the training set but are absent in the current test step. We also define inductive negative edges as test edges which occurred previously in the test set but are not present at the current step. Lastly, we propose a baseline EdgeBank relying solely on memorizing past edges (essentially a hashtable of seen edges). In the plot above, we see that by changing the negative edges for evaluation, the average performance of existing TGL methods reduces significantly in the historical and inductive setting when compared to the standard setting. EdgeBank is also a surprisingly strong baseline for th

https://medium.com/@shenyanghuang1996/towards-bette
cdb8bb1e24e9?source=post_page-----d28d1640dbf2---------



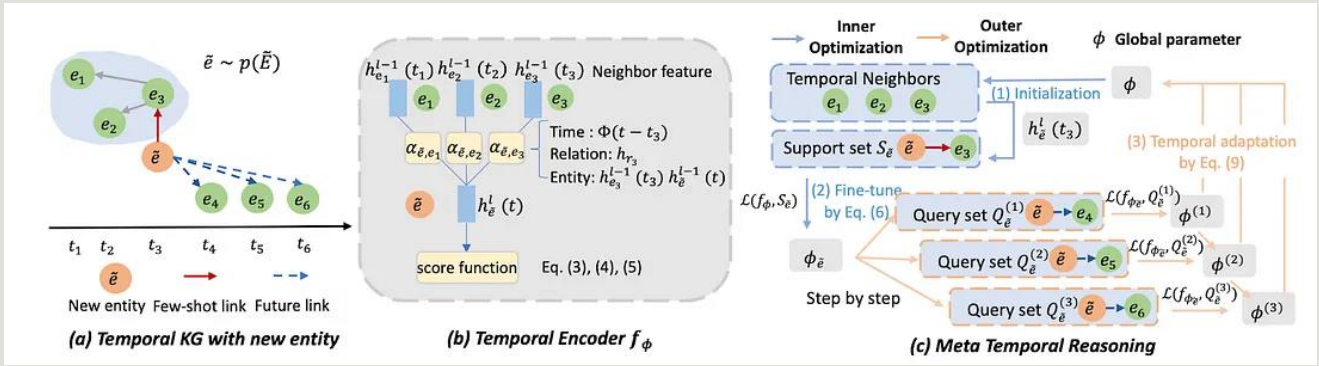
Temporal Knowledge Graphs

In the realm of Knowledge Graphs (KGs), temporal setup is slightly different from the homogeneous world, i.e., timestamped graph snapshots are not that common. Instead, some (or all) triples have an accompanying (start time, end time) pair of attributes denoting the time frame when a given fact was true. Triples thus become quintuples, or, in Wikidata, time attributes become qualifiers of a more general statement (main triple + multiple key-value qualifiers), and statements form so-called hyper-relational KGs.

For example, `(President of the French republic, office holder, Nicolas Sarkozy, 2007, 2012)` is a quintuple describing the time period where Nicolas Sarkozy was the President of the French republic. Alternatively, there can be only one timestamp per triple (forming quadruples). The most common prediction task is scoring head/tail prediction given the time attributes, e.g. , `(President of the French Republic, office holder, ???, 2007, 2012)` — this can be considered as a particular case of the hyper-relational link prediction where qualifiers are only dateTime literals. A classic example of temporal KG completion model is TNTComplex (ICLR 2020).

Temporal Knowledge Graphs

Wang et al. addressed the task of few-shot link prediction over temporal + dynamic graphs where the edges have timestamps and new nodes might appear at later timesteps (Incremental graphs as to the above classification by Krause et al.). The few-shot scenario makes the task even more challenging — we only have access to a limited number of training and inference points (usually, <5) to reason about the queries link. Here, the authors propose MetaTKGR, a meta-learning based approach that builds representations of new nodes by aggregating features of existing nodes within a certain delta t temporal neighborhood. The scalar difference between timestamps is vectorized via the Fourier transform.



Temporal Knowledge Graphs

There were surprisingly few temporal KG papers in this year's top ML conferences: TILP (Xiong et al. 2023) on deriving temporal rule learning competitive with neural methods, and theory work by Chen and Wang to measure expressiveness of temporal GNNs. In fact, the most interesting (to me) papers on this topic were found at the TGL workshop at NeurIPS'23 (one more reason for you to follow the venue!), e.g., predicting future time intervals by Pop and Kostylev, or identifying leakages in standard benchmarking datasets by Pan et al. Finally, I'd outline the Unified Urban KG (UUKG) by Ning et al as a fresh look on temporal KG datasets that actually make practical sense and a use-case — modeling transportation flows in the city.

UUKG illustrates the biggest problem of the shrinking temporal KG community — the lack of practically important tasks and datasets where it would be possible to demonstrate the utility of the data modeling paradigm in real-world tasks. That is, adding 1% of MRR/Hits@10 on 10-year old KG embedding benchmarks is rather useless these days compared to the successes of Geometric Deep Learning in biology or materials science (or compared to LLMs, but that's a story for another day). Hopefully, we will see more UUKG-like practically useful datasets.

Perhaps another adjacent area where temporal KGs might make a difference is heterogeneous graphs (that usually have typed edges) that are much more used in industry. For example, the recent RelBench (Relational Deep Learning Benchmark) formulates a temporal prediction problem over relational databases that can be easily converted to KGs or hypergraphs.

TGB TKG

TKG Methods. Most TKG forecasting methods utilize a discrete time representation, except for [22]. Some methods integrates the message-passing paradigm from static graphs [60, 50] with sequential techniques [30, 39, 20, 21, 37, 42]. Other approaches combine Reinforcement Learning with temporal reasoning for future link prediction [38, 63]. Rule-based methods [44, 32, 49, 45, 41] employ strategies to learn temporal logic rules while others [72, 67, 71] combines a blend of different methodologies. More details are in Appendix G.1. For TKG forecasting, common benchmark datasets include YAGO [48], WIKI [34, 29], GDELT [35] and the Integrated Crisis Early Warning System (ICEWS) dataset [6]. However, these datasets are orders of magnitude smaller than our TKG datasets in number of nodes, edges and timestamps. In tkgl-icews, we include the full ICEWS dataset [6] spanning 28 years when comparing to prior versions containing only one or a few years [12, 29, 9]. Similarly, our tkgl-wikidata dataset is orders of magnitude larger than the existing WIKI dataset [13, 40] in size of nodes, edges and timestamps.

Spatiotemporal Graphs and Graph Deep Learning for Time Series Processing

In the temporal graph learning community, the term spatiotemporal graph has been often used to indicate a graph with fixed topology and node features that change over time, usually at discrete time steps corresponding to regularly sampled observations. More recently the problem of processing data with such a structure is being considered from a different perspective, i.e., by considering the dynamic node feature as time series and edges as functional dependencies among sequences of observations (Cini et al. 2023, Ming et al. 2023). From this perspective, which significantly deviates from many of the settings discussed elsewhere in this blog post, spatiotemporal graph-based representations allow for processing collections of correlated time series by taking advantage of the architectural biases typical of graph neural networks.

Such sets of correlated time series can be generated by sensors, either physical or not. An example is in the traffic domain, where time series might correspond to readings of sensors measuring the number of vehicles passing by at a crossroads. Each sensor will correspond to a different node and an adjacency matrix can be obtained, for instance, by joining with an edge only those sensors directly connected by a road segment. Besides traffic forecasting (Li et al. 2018, Yu et al. 2018), these representations have been used in a wide range of time series processing applications ranging from air quality monitoring (Chen et al. 2021) and energy analytics (Cini et al. 2023) to biomedical data processing (Zhang et al. 2022) and financial time series analysis (Matsunaga et al. 2019).

Spatiotemporal Graphs and Graph Deep Learning for Time Series Processing

To process this data, the standard message-passing framework needs to be updated to handle sequences of observations coming from the neighborhood of each node. This can easily be done by replacing the proper operators (i.e., the message and update functions) with operators able to process the data along the temporal dimension, e.g., recurrent cells (Seo et al. 2018), spatiotemporal convolutions (Wu et al. 2019) and attention-based architectures (Marisca et al. 2022). The resulting models are known as spatiotemporal graph neural networks (STGNNs) and there has been a large amount of research dedicated to coming up with effective architectures (see Ming et al. 2023). One of the main advantages of using STGNNs is that the same set of parameters can be used to forecast any subset of time series, while taking dependencies into account throughout the processing. This is a massive advantage over standard multivariate time series forecasting models that usually would have to forecast each time series separately or give up parameter sharing. Hybrid STGNNs, with some time-series-specific parameters, can also be considered and, as we have shown in a recent NeurIPS paper, often outperform models where all parameters are shared. Besides the model architecture, graph-based representations, as shown by Zambon et al., can also help in assessing the optimality of a forecasting model by focusing the spatiotemporal correlation analysis to interconnected nodes.

Several challenges are inherent to the field, starting from dealing with irregularly sampled time series and missing data; indeed, both are quite common phenomena when dealing with actual cyber-physical systems. Luckily, graph-based models are useful in this context as well, for example allowing to condition the reconstruction on observations at neighboring sensors. Scalability is another major concern as differently from standard GNNs, message-passing is often performed w.r.t. each time step. Existing scalable architectures mostly rely on subsampling and/or pre-computed node features. When no prior relation information is available, the challenge becomes that of learning a latent graph directly from the time series. The problem has been tackled, for example, by directly learning an adjacency matric (e.g., Wu et al. 2019) or, under a probabilistic framework, by relying on reparametrization tricks and score-based estimators.

Since this topic was not covered in last year's blog post, the objective here was to give a short overview of the settings and the problems that can be modeled. Many directions are currently being explored, from graph state-space models and graph Kalman filters to diffusion-based and continuous space-time models. If you are interested in knowing more and/or using these models in practice, we recently released a comprehensive tutorial paper on the topic. You can also check out Torch Spatiotemporal (tsl), our library to build graph-based time series processing pipelines.

Causality-Aware Temporal Graph Learning

One reason why temporal graph learning is interesting is that — depending on the data at hand — it requires different perspectives. As an example, consider temporal graphs data with high-resolution (possibly continuous) time stamps. In such data, discrete-time graph learning techniques that utilize sequences of snapshot graphs require a coarse-graining of time, where each snapshot consists of edges occurring in a certain time interval. This coarse-graining allows to generalize (static) graph learning techniques to time series data. But it introduces a major issue: Each snapshots discards information on the temporal order in which edges occurred, which is the foundation of causal or time-respecting paths (Kempe et al. 2000). Like paths in static graphs, time-respecting paths are important since they tell us which nodes can causally influence each other indirectly.

Causality-Aware Temporal Graph Learning

Several works have shown that — due to the arrow of time — the causal topology of temporal graphs, i.e. which nodes can possibly causally influence each other via time-respecting paths, strongly differs from their static counterparts, with interesting implications for epidemic spreading (Pfitzner et al. 2013), diffusion speed (Scholtes et al. 2014), node centralities (Rosvall et al. 2014), or community detection (Lambiotte et al. 2019). Can we make deep learning methods aware of patterns in the causal topology of temporal graphs? Advances presented at this year show that this can be achieved based on models that generalize commonly used static representations of temporal graphs. Consider a weighted time-aggregated graph, where a (directed) edge (a,b) with weight five captures that (a,b) occurred five times in a temporal graph. Such a weighted, time-aggregated graph is illustrated in the bottom of panel 2 in the figure below.

Each edge in the temporal graph is a time-respecting path with length one. A weighted time-aggregated graph thus corresponds to a first-order model for the causal topology of a temporal graph, which captures time-respecting paths of length one. It neglects the temporal ordering of edges, since we only count how often each edge occurred. A line graph transformation enables us to generalize this idea to causality-aware models that facilitate temporal graph learning: We simply replace edges in the first-order graph by nodes in a second-order graph, i.e. we turn edges (a,b) and (b,c) into nodes “a→b” and “b→c”, respectively. In the resulting second-order graph (see the top graph in panel 2 in figure), we can use edges to represent time-respecting paths of length two, i.e. edge (a→b, b→c) indicates that a causally influence c via b. However, the reverse order of edges are not included. If the edges occur in reverse order, we do not include (a→b, b→c). Importantly, such a second-order graph is sensitive to the temporal ordering of edges, while a first-order graph is not! In Scholtes, 2017, this is generalized to higher orders, which yields k-th order De Bruijn graph models for the causal topology of temporal graphs.

Qarkaxhija et al. have shown that neural message passing in such higher-order De Bruijn graphs yields a causality-aware graph neural network architecture for temporal graphs. Building on these De Bruijn Graph Neural Networks (DBGNN), in a poster at this year’s TGL workshop, Heeg and Scholtes address the challenge to predict temporal betweenness and closeness centralities of nodes. Since they are influenced by the arrow of time, temporal node centralities can drastically differ from static centralities. Moreover, it is costly to compute them! This is addressed by training a DBGNN model on a first time interval of a temporal graph, then using the trained model to forecast temporal centralities in the remaining data. The overall approach is illustrated above. Empirically results are promising and showcased the potential of causality-aware graph learning. We also hope to see more attention from the community in learning causal structure on temporal graphs in 2024.

Interested in causality-aware temporal graph learning? Then there’s good news! The techniques above are implemented in the Open Source library pathpyG, which builds on PyG. There is an introductory video and a tutorial available. A recorded talk given in the temporal graph reading group provides an in-depth introduction of the underlying research.

Libraries and Datasets

In 2023 saw a significant push towards standardized libraries and benchmarks for temporal graph learning. TGB provides an open and standardized benchmark for node and link level tasks. DyGLib is a library which includes standard training pipelines, extensible coding interfaces, and comprehensive evaluation strategies for temporal graph learning. Zhang et al. introduced the novel concept of *Live Graph Lab*, providing live graphs according to blockchain transactions. With a set of tools for downloading, parsing, cleaning, and analyzing blockchain transactions, Live Graph Lab offers researchers the opportunity to extract update to date temporal graph data any time and use it for analysis or testing. Zhou et al. noticed that node memory used in TG models favors small batch sizes and needs to be maintained synchronously across all trainers. Therefore, they proposed DistTGL, an efficient and scalable solution to train memory-based TGNNs on distributed GPU clusters. Enabling multi-GPU training on large datasets is an important direction to deploy TG models on large datasets. We present an updated list of libraries and benchmarks for temporal graph learning:

- TGB website and pypi install
- DyGLib Github
- Live Graph Lab website and dataset
- DistTGL Github
- Torch Spatiotemporal (TSL) website and Github
- pathpyG website and Github
- RelBench website and Github
- Pytorch Geometric Temporal
- TGL paper and Github
- DGB pypi install, paper and datasets
- Chartalist Blockchain Network website, paper and Github
- TKG Forecasting Evaluation paper and Github

Libraries and Datasets

Poursafaei et al. [33] collected six novel datasets for link prediction on continuous-time dynamic graphs while proposing more difficult negative samples for evaluation. In comparison, we curated seven novel temporal graph datasets spanning both edge and node level tasks for realistic evaluation of machine learning on temporal graphs. Yu et al. [52] presented DyGLib, a platform for reproducible training and evaluation of existing TG models on common benchmark datasets. DyGLib demonstrates the discrepancy of the model performance across different datasets and argues that diverse evaluation protocols of previous works caused an inconsistency in performance reports. Similarly, Skarding et al. [39] provided a comprehensive comparative analysis of heuristics, static GNNs, discrete dynamic GNN, and continuous dynamic GNN on dynamic link prediction task. They showed that dynamic models outperforms their static counterparts consistently and heuristic approaches can achieve strong performance. In all of the above benchmarks, the included datasets only contain a few million edges. In comparison, TGB datasets are orders of magnitude larger in scale in terms of number of nodes, edges and timestamps. TGB also includes both node and edge-level tasks. Huang et al. [18] collected a novel dynamic graph dataset for anomalous node detection in financial networks and compared the performance of different graph anomaly detection methods. In this work, TGB datasets have more edges and timestamps while covering both edge and node tasks.

Applications of Temporal Knowledge Graph

Temporal Knowledge Graph Reasoning TKGRL methods are widely used in temporal knowledge graph reasoning (TKGR) tasks which automatically infers new facts by learning the existing facts in the KG. TKGR usually has three subtasks: entity prediction, relation prediction, and time prediction. Entity prediction is the basic task of link prediction, which can be expressed as two queries (?, r, t, τ) and (h, r, ?, τ). Relation prediction and time prediction can be expressed as (h, ?, t, τ) and (h, r, t, ?), respectively. TKGR can be divided into two categories based on when the predictions of facts occur, namely interpolation and extrapolation. Suppose that a TKG is available from time τ0 to τT . The primary objective of interpolation is to retrieve the missing facts at a specific point in time τ (τ0 ≤ τ ≤ τT). This process is also known as temporal knowledge graph completion (TKGC). On the other hand, extrapolation aims to predict the facts that will occur in the future (τ ≥ τT) and is referred to as temporal knowledge graph forecasting. Several methods have been proposed for Temporal Knowledge Graph Completion (TKGC) including transformation-based, decomposition-based, graph neural networks-based, capsule Network-based, and other geometric methods. These techniques aim to address the problem of missing facts in TKGs by leveraging various mathematical models and neural networks. In contrast, predicting future facts in TKGs requires a different approach that can model the temporal evolution of the graph. Autoregression-based, temporal point process-based, and few-shot learning methods are commonly used for this task. Interpretability-based methods are used to increase the reliability of prediction results. These techniques provide evidence to support predictions, helping to establish trust and improving the overall quality of predictions made by the model. To further enhance the performance of TKGRL, semantic augmentation technology can be employed to improve the quality and quantity of semantic information of TKGs. Utilizing entity and relation names, as well as textual descriptions of fact associations, can enrich their representation and promote the development of downstream tasks of TKGs. In addition, large language models 20 (LLMs) for natural language processing (NLP) can facilitate the acquisition of rich semantic information about entities and relations, further augmenting the performance of TKGRL models. 4.2. Entity Alignment Between Temporal Knowledge Graphs Entity alignment (EA) aims to find equivalent entities between different KGs, which is important to promote the knowledge fusion between multi-source and multi-lingual KGs. Defining G1 = (E1, R1, T1, F1) and G2 = (E2, R2, T2, F2) to be two TKGs, S = {(e1i , e2j)|e1i ∈ E1, e2j ∈ E2} is the set of alignment seeds between G1 and G2. EA seeks to find new alignment entities according to the alignment seeds S. The methods of EA between TKGS mainly adopt the GNN-based model. Currently, exploring the entity alignment (EA) between Temporal Knowledge Graphs (TKGs) is an active area of research. TEA-GNN [81] was the first method to incorporate temporal information via a time-aware attention Graph Neural Network (GNN) to enhance EA. TREA [82] utilizes a temporal relational attention GNN to integrate relational and temporal features of entities for improved EA performance. STEA [7] identifies that the timestamps in many TKGs are uniform and proposes a simple GNN-based model with a temporal information matching mechanism to enhance EA. Initially, the structure and relation features of an entity are fused together to generate the entity embedding. Then, the entity embedding is updated using GNN aggregation from neighborhood. Finally, the entity embedding is obtained by concatenating the embedding of each layer of the GNN. STEA not only updates the representation of entities but also calculates time similarity by considering associated timestamps. The method combines both the similarities of entity embeddings and the similarities of entity timestamps to obtain aligned entities. Overall, STEA offers an effective way of improving entity representation in TKGs and provides a reliable solution for aligning entities over time. 4.3. Question Answering Over Temporal Knowledge Graphs Question answering over KG (KGQA) aims to answer natural language questions based on KG. The answer to the question is usually an entity in the KG. In order to answer the question, one-hop or multi-hop reasoning is required on the KG. Question answering over TKG (TKGQA) aims to answer temporal natural language questions based on TKG, the answer to the question is entity or timestamp in the TKG, and the reasoning on TKG is more complex than it on KG. Research on TKGQA is in progress. CRONKGQA [59] release a new dataset named CRONQUESTIONS and propose a model combining representation of TKG and question for TKGQA. It first uses TComplEx to obtain the representation of entities and timestamps in the TKG, and utilizes BERT [15] to obtain their representations in the question, then calculates the scores of all entities and times, and finally concatenated the score vectors to obtain the answer. 21 TSQA [63] argues existing TKGQA methods haven’t explore the implicit temporal feature in TKGs and temporal questions. It proposes a time sensitive question answering model which consists of a time-aware TKG encoder and a time-sensitive question answering module. The time-aware TKG encoder uses TComplEx with time-order constraints to obtain the representations of entities and timestamps. The time-sensitive question answering module first decomposes the question into entities and a temporal expression. It uses the entities to extract the neighbor graph to reduce the search space of timestamps and answer entities. The temporal expression is fed into the BERT to learn the temporal question representations. Finally, entity and temporal question representations are combined to estimate the time and predict the entity with contrastive learning.

TGB dataset

tgbl-wiki. This dataset stores the co-editing network on Wikipedia pages over one month. The network is a bipartite interaction network where editors and wiki pages are nodes, while one edge represents a given user edits a page at a specific timestamp. Each edge has text features from the page edits. The task for this dataset is to predict with which wiki page a user will interact at a given time. tgbl-review. This dataset is an Amazon product review network from 1997 to 2018 where users rate different products in the electronics category from a scale from one to five. Therefore, the network is a bipartite weighted network where both users and products are nodes and each edge represents a particular review from a user to a product at a given time. Only users with a minimum of 10 reviews within the aforementioned time interval are kept in the network. The considered task for this dataset is to predict which product a user will review at a given time. tgbl-coin. This is a cryptocurrency transaction dataset based on the Stablecoin ERC20 transactions dataset [37]. Each node is an address and each edge represents the transfer of funds from one address to another at a time. The network starts from April 1st, 2022, and ends on November 1st, 2022, and contains transaction data of 5 stablecoins and 1 wrapped token. This duration includes the Terra Luna 6 crash where the token lost its fixed price of 1 USD. The considered task for this dataset is to predict with which destination a given address will interact at a given time. tgbl-comment. This dataset is a directed reply network of Reddit where users reply to each other's threads. Each node is a user and each interaction is a reply from one user to another. The network starts from 2005 and ends at 2010. The considered task for this dataset is to predict if a given user will reply to another one at a given time. tgbl-flight. This dataset is a crowd sourced international flight network from 2019 to 2022. The airports are modeled as nodes, while the edges are flights between airports at a given day. The node features include the type of the airport, the continent where the airport is located, the ISO region code of the airport as well as its longitude and latitude. The edge feature is the associated flight number. In this dataset, our task is to predict whether a flight will happen between two specific airport on a future date. This is useful for foreseeing potential flight disruptions such as cancellation and delays. For instance, during the COVID-19 pandemic, many flight routes were cancelled to combat the spread of COVID-19. In addition, the prediction of global flight network is also important for studying and forecasting the spread of disease such as COVID-19 to new regions, as shown in [3, 11]. tgbn-trade. This is the international agriculture trading network between nations of the United Nations (UN) from 1986 to 2016. Each node is a nation and an edge represents the sum trade value of all agriculture products from one nation to another one. As the data is reported annually, the time granularity of the dataset is yearly. The considered task for this dataset is to predict the proportion of agriculture trade values from one nation to other nations during the next year. tgbn-genre. This is a bipartite and weighted interaction network between users and the music genres of songs they listen to. Both users and music genres are represented as nodes while an interaction specifies a user listens to a music genre at a given time. The edge weights denote the percentage of which a song belongs to a certain genre. The dataset is constructed by cross referencing the songs in the LastFM-song-listens dataset [24, 15] with that of music genres in the million-song dataset [2]. The LastFM-song-listens dataset has one month of who-listens-to-which-song information for 1000 users and the million-song dataset provides genre weights for all songs in the LastFM-song-listens dataset. We only retain genres with at least 10% weights for each song that are repeated at least a thousand times in the dataset. Genre names are cleaned to remove typos. Here, the task is to predict how frequently each user will interact with music genres over the next week. This is applicable to many music recommendation systems where providing personalized recommendation is important and user preference shifts over time. tgbn-reddit. This is a users and subreddits interaction network. Both users and subreddits are nodes and each edge indicates that a user posted on a subreddit at a given time. The dataset spans from 2005 to 2019. The task considered for this dataset is to learn the interaction frequency towards the subreddits of a user over the next week. tgbn-token. This is a user and cryptocurrency token transaction network. Both users and tokens are nodes and each edge indicates the transaction from a user to a token. The edge weights indicate the amount of token transferred and considering the disparity between weights, we normalized the edge weights using logarithm. The goal here is to predict how frequently a user will interact with various types of tokens over the next week. The dataset is extracted and curated from this source [37].

Temporal Graph Benchmark

Temporal Graph Benchmark (TGB) was presented recently, including a collection of challenging and diverse benchmark datasets for realistic, reproducible, and robust evaluation for machine learning on temporal graphs. TGB provides a pypi package to automatically download and process nine datasets from five distinct domains with up to 72 million edges and 30 million timestamps. TGB also provides standardized evaluation motivated by real applications.

TGB includes both link and node level tasks and an extensive empirical comparison of state-of-the-art TG models on all datasets. The first task is the dynamic link property prediction task which predicts the property (often existence) of a link between a pair of nodes at a future time. In TGB, this task is modeled as a ranking problem and evaluated with the filtered Mean Reciprocal Rank (MRR) metric. Results show that model rankings vary significantly across datasets with different ratios of test set edges which are never observed during training. In addition, model performance deteriorates as more negative samples (non-existence edges) are used in the evaluation. Interestingly, the extent of performance drop varies across models as well.

In the dynamic node property prediction task, the goal is to predict the property of a node at a given time. More specifically we focus on the node affinity prediction task which models how the user preference towards different items shift over time. Here, we use the Normalized Discounted Cumulative Gain of the top 10 items (NDCG@10) to compare the relative order of the predicted items to that of the ground truth. Interesting, we found that single heuristics outperform existing TG models and this highlights the need for more models focusing on node level tasks in the future. The TGB leaderboard is public and you are welcome to submit your model via a google form. For more details, see the TGB blog post by the authors of this blog.

dataset selection and evaluation protocol

Dynamic Link prediction: A natural problem for a graph is to predict if there is a link (with label r in case of a KG) between two nodes v and u. In the dynamic case, one may be interested in predicting if such a link existed at time t in the past or if it will appear sometime in the future. An example of this would be to predict whether Donald Trump will visit China in the next year or not and we query for this link in a KG. Example applications include temporal KG completion, friend recommendation, finding biological connections among species, and predicting obsolete facts in a KG. The most common evaluation metrics for this task include: AOC (area under ROC curve), corresponding to the probability that the predictor gives a higher score to a randomly chosen existing link than a randomly chosen nonexistent one, GMAUC, a geometric mean of AOC, and PRAUC (area under precision-recall curve) - see, for example, Chen et al. (2019a). Another metric named error 42 Representation Learning for Dynamic Graphs rate was considered in Chen et al. (2019a) corresponding to the ratio of the number of mispredicted links to the total number of truly existing links. Dynamic Entity/relation prediction: One of the fundamental problems in KGs is to predict missing entities or relations. It is a classical problem where we want to predict either a missing head entity (?,r, u), a missing tail entity (v,r, ?), or a missing relation (v, ?, u). In the context of a DTDG, one may want to predict the missing entity or relation in the next snapshot. In the case of a CTDG, one may want to predict the missing entity or relation at a specific timestamp t (e.g., (?,r, v, t)). An example mentioned by Trivedi et al. (2017) is to predict who Donald Trump will mention next? Leblay and Chekol (2018) and Dasgupta et al. (2018) considered the task of predicting a missing entity in the temporal case. They rank all entities that can potentially be the missing entity and then find the rank of the actual missing entity. They used mean rank (MR), and the percentage of cases where the actual missing entity is ranked among the top K (known as Hit@K) to compare the quality of the results. In addition to the above metrics, Garc´ıa-Dur´an et al. (2018) and Goel et al. (2020) also computed mean reciprocal rank (MRR). MRR is generally reported under two settings: raw and filtered (see Bordes et al. (2013) for the details). Recommender systems: The design of dynamic recommender systems is an important applied dynamic graph problem faced by a myriad of e-commerce companies in online retail, video streaming, and music streaming, to name a few examples. In a dynamic recommendersystem problem, we have a set of users, a set of items, and a set of timestamped interactions between users and items, and we seek to recommend items to users based on their current tastes (Wu et al. (2017); Kumar et al. (2018b)). The key is that tastes may exhibit cyclic or trend behavior which should be adapted to or anticipated inasmuch as this is possible. As a coarse approximation, we can view a dynamic recommender-system problem as one of dynamic link prediction (or dynamic entity prediction) and attempt to recommend to users the items they are likely to autonomously choose. However, a more fine-grained analysis reveals several complications present in recommender systems that may be absent in other dynamic link prediction problems. First, the actual output of a recommender system for a given user is a sequence of slates of recommended items. If the items on an output slate are highly similar, then their utility to the user may be strongly correlated, so that there is a non-negligible risk of the slate being useless to the user. This risk is mitigated with a more diverse slate, even if the diverse slate has a lower sum of expected utilities than the uniform slate (see Kaminskas and Bridge (2017))4 . Second, from the point of view of an e-commerce company, the purpose of a recommender system is typically to maximize profit in the long term. Therefore, it may be desirable to recommend to the user items in which the user has no immediate interest, but which are expected to cause the user to purchase profitable items or click on profitable advertisements. From this perspective, the design of a dynamic recommender system can be seen as a reinforcement learning problem on a dynamic graph. One seeks to dynamically control an evolving graph to maximize some profit-related objective. Time Prediction: For dynamic graphs, besides predicting which event will happen in the future, an interesting problem is to also predict when that event will happen. As compared to other tasks, this task only exists for dynamic networks. For instance, we saw the example 4. This reasoning resembles Markowitz's portfolio theory (Markowitz (1952)). 43 Kazemi, Goel, Jain, Kobyzev, Sethi, Forsyth, and Poupart in Section 5.1 of predicting when Bob will visit Montreal. A similar time prediction problem is the temporal scoping problem in KG completion where the goal is to predict missing timestamps (e.g., answering queries such as (v,r, u, ?), where v is known to have had a relation r with u in the past). Sun et al. (2012) and Trivedi et al. (2017, 2019) used the mean absolute error between the predicted time and ground truth to measure the quality of the results. Dasgupta et al. (2018) ordered the predicted timestamps in the decreasing order of their probabilities and selected the rank associated with the correct timestamp. They computed the mean rank (MR) to compare the results. Node classification: Node classification is the problem of classifying graph nodes into different classes. An example of a node classification problem is to predict the political affiliation of the users of a social network based on their attributes, connections, and activities. In particular, one may be interested in making such predictions in a streaming scenario where the classification scores keep updating as new events happen or as a user activity is observed. Node classification is often studied under two settings: transductive and inductive. In the transductive setting (also known as semi-supervised classification), given the labels of a few nodes, we want to predict the labels of the other nodes in the graph. In the inductive setting, the label is to be predicted for new nodes that have not been seen during training. The problem of node classification becomes challenging in the dynamic case as the distribution of the class labels may change over time. There are not many publicly-available datasets for dynamic node classification. Pareja et al. (2019) used Elliptic a network of bitcoin transactions, for temporal node classification. Sato et al. (2019) considered a dataset of face to face proximity for temporal node classification. They used the SIR model for node classification, which is a popular framework to model the spread of epidemics (Hethcote (2000)). At each timestamp, each node can be one of three possible states: susceptible (S), infectious (I), and recovered (R). Classification accuracy is a widely used metric for this task. For datasets where the task is a multi-class classification, micro-F1 and macro-F1 scores are also used to measure the performance (Cui et al. (2018)). Graph classification: Graph classification is the problem of classifying the whole graph into one class from a set of predefined classes. This task can be useful in domains like bioinformatics and social networks. In bioinformatics, for instance, one important application is the protein function classification where proteins are viewed as graphs. As another example, Taheri et al. (2019) modeled activity state classification for a troop of GPS-tracked baboons as a dynamic graph classification problem where the labels are different activity states such as sleeping, hanging-out, coordinated non-progression, and coordinated progression. Common graph classification benchmarks include COLLAB (Yanardag and Vishwanathan (2015)), PROTEINS (Borgwardt et al. (2005)) and D&D (Dobson and Doig (2003)). The performance is typically measured in terms of classification accuracy. Network clustering: Network clustering or detecting communities in graphs such as social networks, biological networks, etc. is an important problem. A network cluster/community typically refers to a subset of nodes that are densely connected, but loosely connected to the rest of the nodes. One challenge in community detection for dynamic graphs is that one needs to model how communities evolve. A common performance measure used for this task is the overlap between the predicted and the true cluster assignments. Xin et al. (2016) utilize the random walk encoders to find the closely associated nodes for a given node and cluster the global network into overlapping communities. Furthermore, the closely associated nodes 44 Representation Learning for Dynamic Graphs are updated when impacted by dynamic events, giving a dynamic community detection method. Chen et al. (2019b) recast community detection as a node-wise classification problem and present a family of graph neural networks for solving the community detection problem in a supervised setting. Crawford and Milenkovi´c (2018) considered a dynamic communication network between 75 patients and health workers in France with four different clusters representing doctors, administrators, nurses, and patients. Dynamic Question/query answering: The way search engines respond to our questions has evolved in the last few years. While traditionally search engines were aiming at suggesting documents in which the answer to a query question can be potentially found, these days they try to directly answer the question. This has become possible in part due to question answering over KGs (QA-KG). Jia et al. (2018) further extend this work to temporal questions. They defined a temporal question as a question that has a temporal expression (date, time, interval, and periodic events) or temporal signal (before, after, during, etc.) in the question or whose answer is temporal. For QA-KG, the most popular metrics are precision, recall, F1 score, AUC, and accuracy. An example of a temporal question given by Jia et al. (2018) was, "Which teams did Neymar play for before joining PSG?" Hamilton et al. (2018) propose a way of mapping a query formulated as a conjunctive first-order logic formula into a vector representation (corresponding to a query embedding) through geometric operations. The geometric operations leading to query embeddings are jointly optimized with node embeddings such that the query embedding is close in the embedded space to the embeddings of the entities corresponding to the correct answer of the query. For QA-KG, the most popular metrics are precision, recall, F1 score, AUC, and accuracy.

Disease modeling

In the recent COVID-19 pandemic, epidemic modeling is instrumental for understanding the spread of the disease as well as designing corresponding intervention strategies. Human contact networks are in fact temporal graphs. By combining contact graphs with classical compartment based models such as SEIR and SIR, we can more accurately forecast COVID-19 infection curve and go beyond the homogenous mixing assumption (all individuals are equally likely to be in contact with each other).

Chang et al. derived a temporal mobility network from cell phone data and mapped the hourly movement of 98 million people from census block groups (CBGs) to specific points of interest (POIs) in the US. By combining the hourly contact network with SEIR models on the CBG level, they are able to accurately fit real infection trajectory. In particular, the model shows that some 'superspreader' POIs such as restaurants and fitness centers account for a large majority of the infections. Also, differences in mobility between racial and socioeconomic groups lead to different infection rates among these groups. This work showcased the real world potential of utilizing large scale temporal graphs for disease forecasting and informing policies on intervention strategies.

Besides human contact networks, dynamic transportation networks also play an important role in the spread of COVID-19. In recent work by one of the authors, we incorporated daily flight networks into the SEIR model to estimate imported COVID-19 cases. By incorporating flight networks, it is possible for early detection of outbreaks and forecast the impact of travel restrictions. See the blog post by one of the authors for more details.

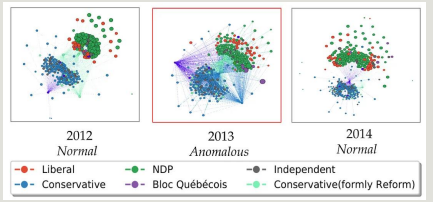
Despite the empirical success of temporal-graph-based disease models, it is also important to answer questions such as "how does the contact network structure impact the spread of disease?" and "what is the best way to modify the contact pattern such that the spread of COVID-19 can be slowed or prevented?" Holme et al. compared the difference in outbreak characteristics between using temporal, static and fully-connected networks on eight network datasets and examined various network structures affecting the spread of the disease. They showed that converting temporal networks into static ones can lead to severe under- or over-estimation of both the outbreak size and extinction time of the disease.

What are the next steps for TGL on epidemic modeling?

1). First, forecasting the entire contact or mobility network snapshot for the immediate future is a crucial challenge. With the predicted structure, we can apply network based SEIR models to estimate the infection curve.

2). Second, defining and understanding the impact of interaction patterns on the contact network is crucial for policy making and interpretability. Analyzing the interplay between graph structures and the infection curve can help us identify the most effective intervention strategies.

Applications: Anomaly Detection



Anomaly detection is a fundamental task in analyzing temporal graphs which identifies entities that deviate significantly from the rest. For example, fraud detection can be modeled as detecting abnormal edges in a transaction network and traffic accident identification can be seen as detecting anomalous events in a traffic network.

There is growing interest in utilizing the representation power of temporal graph networks for anomaly detection. Cai et al. designed an end-to-end structural temporal Graph Neural Network model for detecting anomalous edges, called StrGNN. An enclosing subgraph, a k-hop subgraph centered around an edge, is first extracted based on the edge of interest to reduce computational complexity. A Graph Convolutional Neural Network (GCN) is then used to generate structural embedding from the subgraph. Gated Recurrent Units (GRUs) are then used to capture temporal information. One of the challenges of anomaly detection is the lack of labeled examples. Therefore, Cai et al. proposed to generate "context-dependent" negative edges by replacing one of the nodes in a normal edge and training the model with these negative edges.

When comparing with unsupervised, non-GNN based anomaly detection methods such as SEDANSPOT and AnomRank, GNN based methods can easily incorporate any given attribute and has the potential to achieve stronger performance. However, there are two significant challenges for GNN based approaches.

1). First, how to scale to dynamic graphs with millions of edges and nodes? This is an open question for both the GNN module in extracting the graph features but also the temporal module such as GRUs and transformers in processing long term information.

2). Second, how to produce accurate explanations for the detected anomalies? In real applications, detected anomalies are often verified and then potentially resulting in punitive measures for those detected entities. GNN explainability on dynamic graphs remains an open challenge.

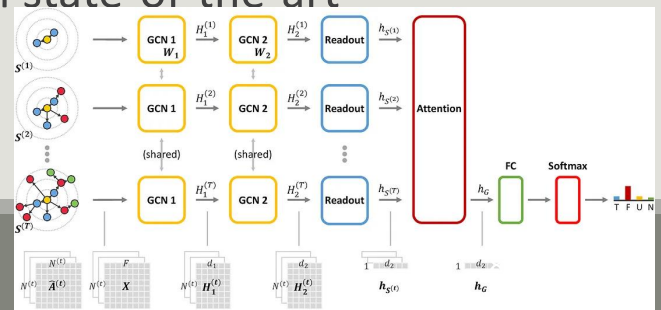
The task of change point detection aims to detect time points in a dynamic graph where the graph structure or distribution deviates significantly from what was observed before. This change can be attributed to external events (such as traffic disruption and COVID-19 related flight restrictions) or simply natural evolution of the dynamic graph. Recent work by one of the authors utilized the eigenvalues of the Laplacian matrix of each graph snapshot to embed the graph structure while applying sliding windows to compare the changes in graph structure in the long and short term. In the above, the proposed Laplacian Anomaly Detection (LAD) method detects a change in the Canadian Member of Parliament (MP) voting network due to increased edges between political parties. This coincides with Justin Trudeau being selected as the Liberal party leader in 2013.

Detecting Misinformation

Misinformation spreads in different patterns and rates when compared to true information (Vosoughi et al.). There has been considerable research studying these network patterns in a static graph while dynamic graph based methods are underexplored (Song et al.). However, in the past year an increased amount of TGL methods were employed for misinformation detection and understanding. For instance, Zhang et al. developed a method based on Temporal Point Processes while Dynamic GCN (DynGCN) and DGNF are dynamic GNN based methods.

The illustration below shows the architecture of DynGCN. They construct graph snapshots with even spacing in time, feed each through GCN layers, and then combine the representations and learn the snapshots' evolution patterns using attention. This is a relatively simpler approach to leverage temporal information compared to some methods discussed above like TGN or CAW, but nonetheless gives better performance than previous state-of-the-art for misinformation detection on the datasets that the authors examined.

Dynamic interaction patterns are shown to be quite informative for misinformation detection (Plepi et al.). With significant recent advances in TGL methods, we can expect novel state-of-the-art misinformation detection methods that incorporate dynamic graphs.



Applications: Adversarial Attacks

Adversarial attacks can target the privacy of customers or affect critical decisions in financial systems. As temporal graph models are deployed to applications such as recommendation systems and fraud detection, it is important to investigate attacks and design defense mechanisms for TG models. Chen et al. proposed the first adversarial attack for dynamic link prediction called Time-aware Gradient Attack (TGA) for discrete time dynamic graphs. TGA rewires a limited number of links from the original network and the most valuable links to the predicted link are determined by the gradient information generated by the TG model.

Recently, Sharma et al. argued that effective attacks on temporal graphs must optimize both edge and time perturbations while preserving the original graph evolution. This is because drastic attacks that disturb the graph evolution would be easily detected by anomaly detection methods. Therefore, Sharma et al. formulated evolution-preserving attacks on discrete-time dynamic graphs as the Temporal Dynamics-Aware Perturbation (TDAP) constraint. TDAP asserts that perturbations added at a given timestamp must only be a small fraction of the actual number of changes with respect to the preceding timestamp. TDAP is shown to preserve the rate of change in both the structure and the embedding spaces. An overview of TDAP is shown in the figure below. Sharma et al. then proposes a novel attack method called Temporal Dynamics-aware Projected Gradient Descent (TD-PGD) which is shown to have a closed-form projection operator under the TDAP constraint. An online version of TD-PGD is also proposed where perturbations can be added in real time. Lastly, it is shown empirically that TDAP-constrained perturbations can indeed evade attacks by embedding-based anomaly detection methods.

Face-to-face interactions

A. Longa, G. Cencetti, B. Lepri, and A. Passerini. An efficient procedure for mining egocentric temporal motifs. Data Mining and Knowledge Discovery, 2022.

Human mobility

G. Mauro, M. Luca, A. Longa, B. Lepri, and L. Pappalardo. Generating mobility networks with generative adversarial networks. EPJ Data Science, 11(1):58, 2022.

S. Gao. Spatio-temporal analytics for exploring human mobility patterns and urban dynamics in the mobile age. Spatial Cognition & Computation, 15(2):86–114, 2015.