INSTITUTE OF TECHNOLOGY OF CAMBODIA

Graduate School of ITC

Master of Data Science

Temporal Graph Learning with Application to Large-Scale Flight Traffic Prediction

Student: Mr. HOR Hang
Advisor: Dr. PHAUK Sokkhey
Co-Advisors: Dr. Gabor BENEDEK (*)
Dr. HAS Sothea
Dr. NEANG Pheak

2023-2024

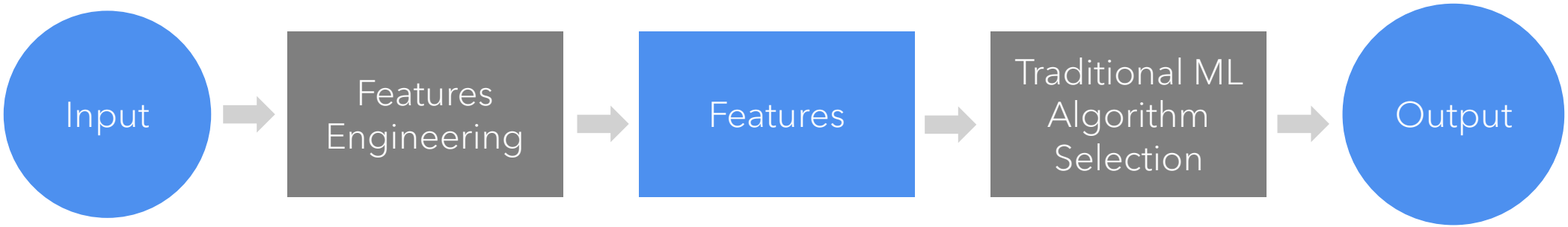
[*] Co-founder / Chief Innovation Officer @Lynx Analytics, Singapore

Contents

Introduction (1)

Traditional approaches

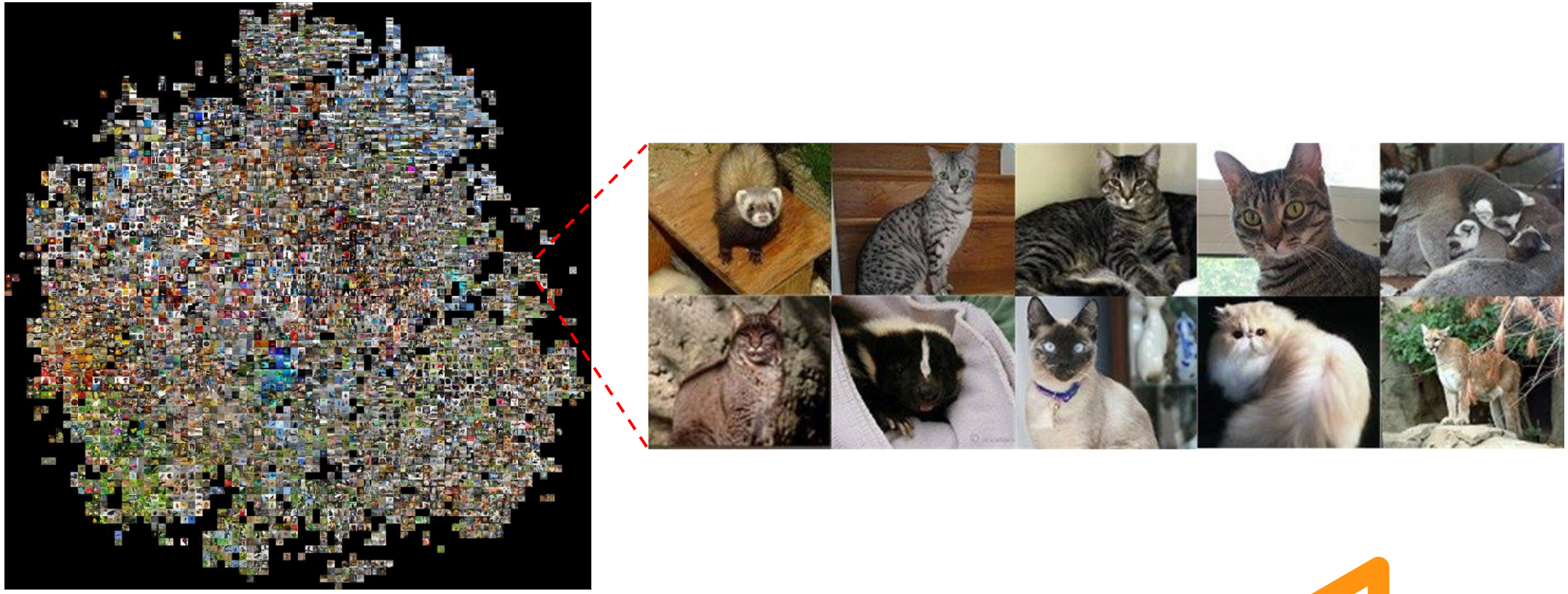
Handcrafted Approach:



Introduction (2)

The struggle

Handcrafted approach failed:

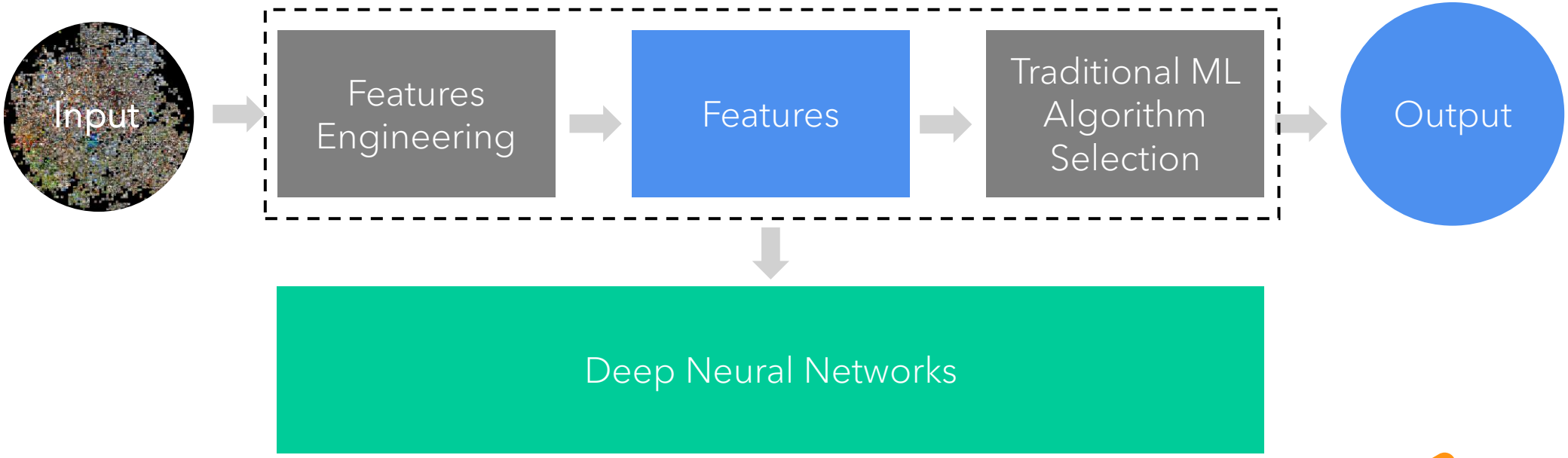


ImageNet [1]
Source: https://cs.stanford.edu/people/karpathy/cnnembed/

Introduction (3)

The rise of new approaches

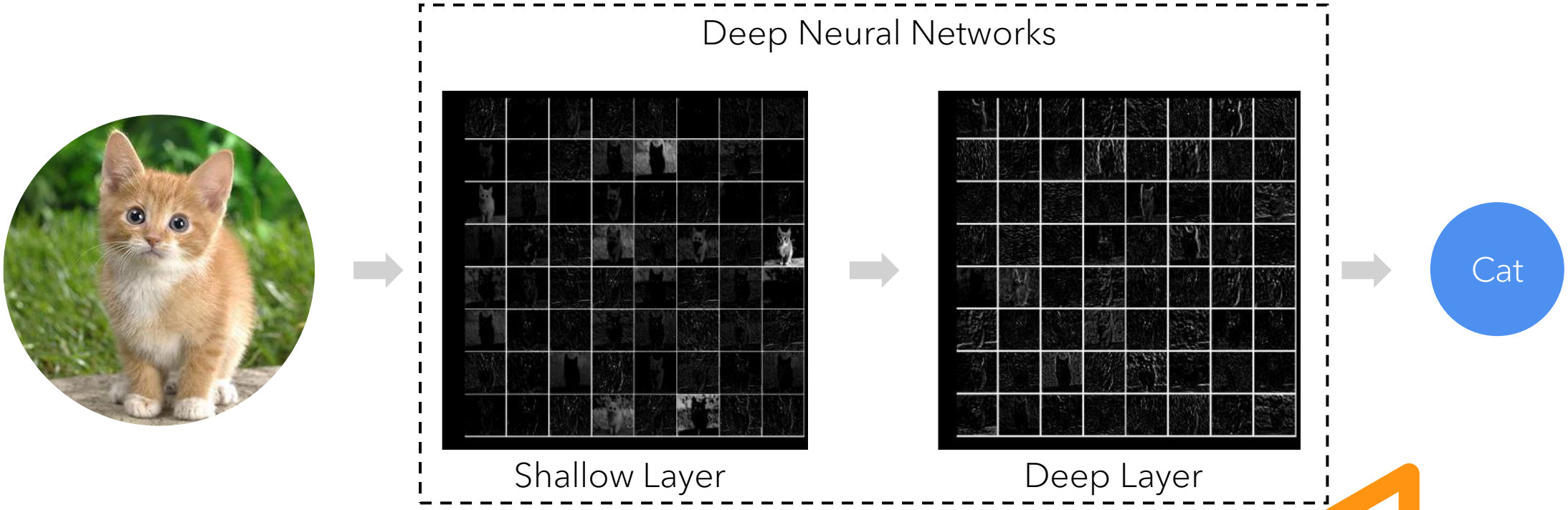
Deep Learning Approach:



Introduction (4)

Into the realm of deep neural network

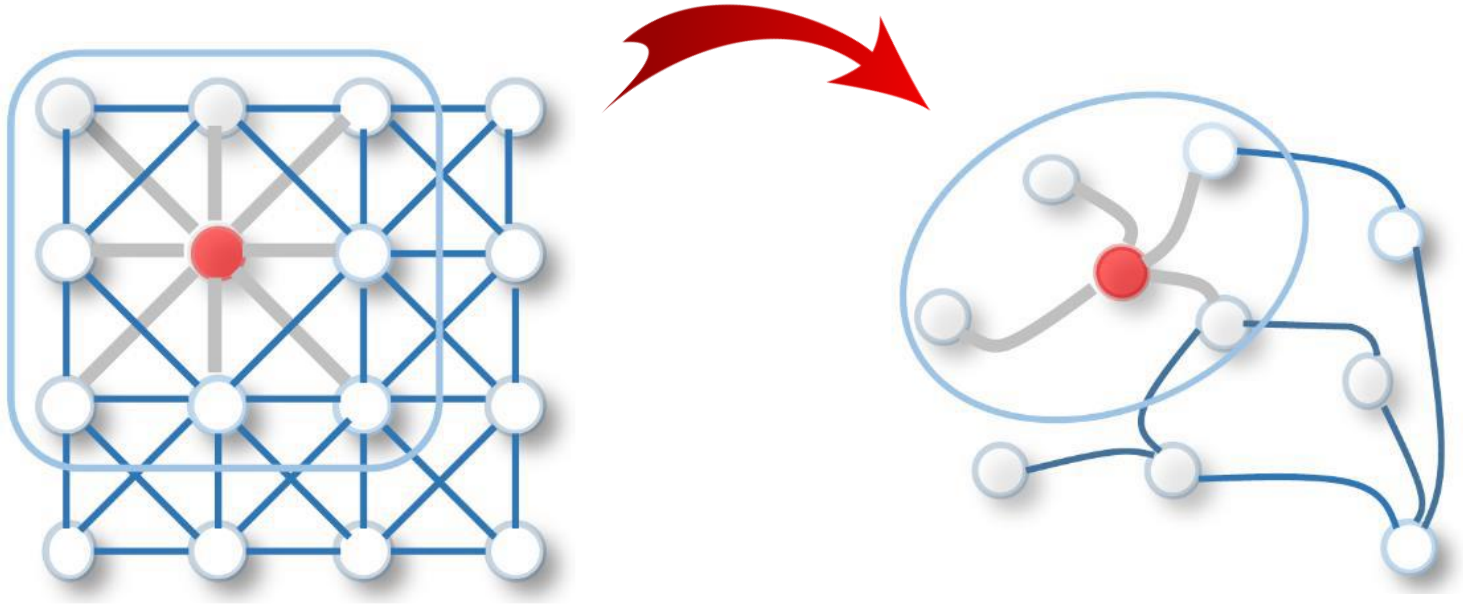
Example:



Introduction (5)

Domain adaptation

From image to graph:



Source: [2]

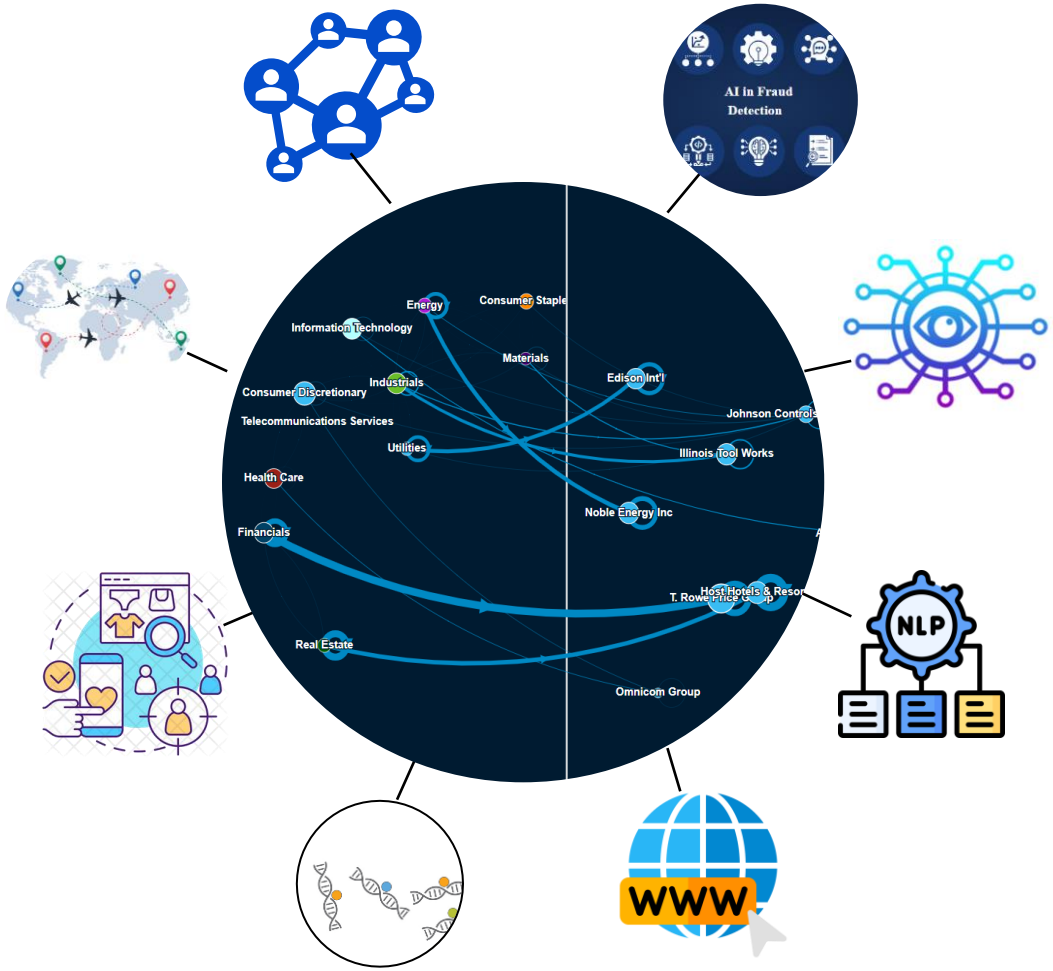
Image Domain
(Kernel of CNN)

Graph Domain
(Neighborhood of GNN)

Introduction (6)

Graph applications

Graphs are everywhere!!!



Introduction (7)

Machine learning on graphs

Graph Modeling

System of interaction modeled as Graph Networks with entities as nodes and interaction as edges

Graph Machine Learning

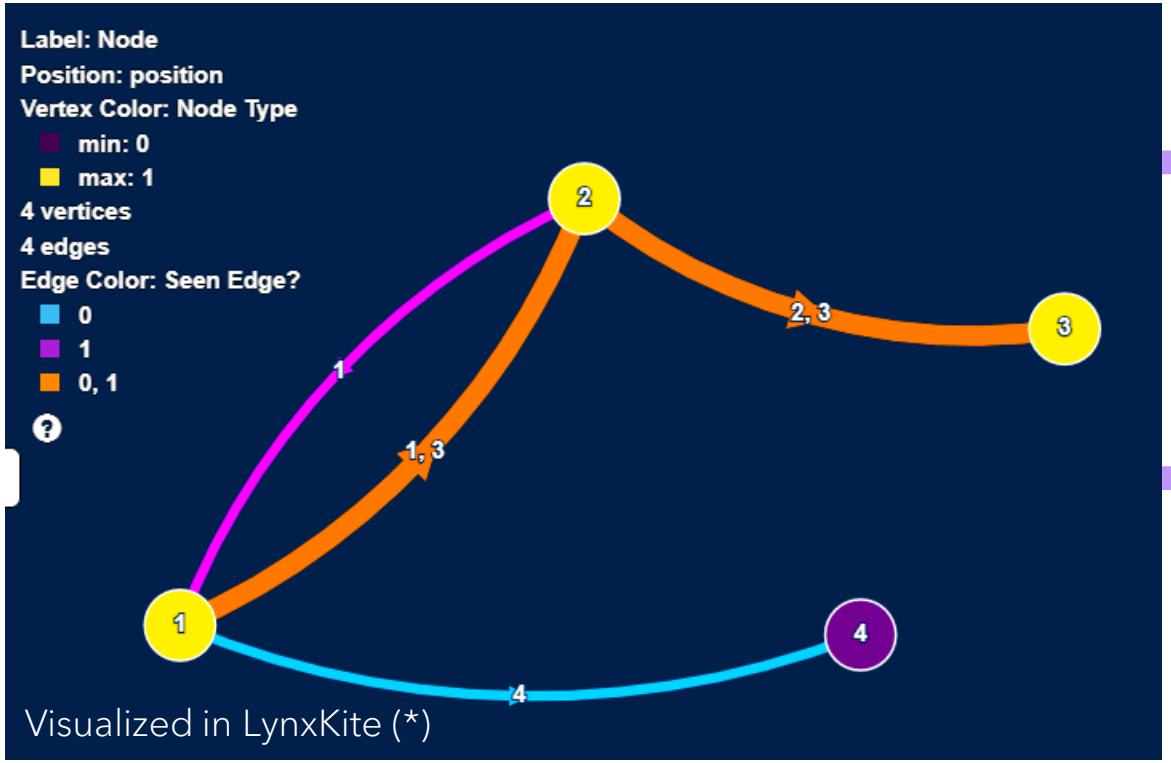
Framework successful in social networks, transportation, recommendation systems

Static Assumption Limitation

Existing methods are limited by static assumptions, not applicable to real-world networks

Real-World Graphs Example

Directed graph with temporal changes in nodes and edges



Visualized in LynxKite (*)

(*) https://try.lynxkite.com/

Introduction (8)

Problem statements

TGB Dataset[15]

67 million edges, a Large-scale crowd-sourced international flight network from 2019 to 2022

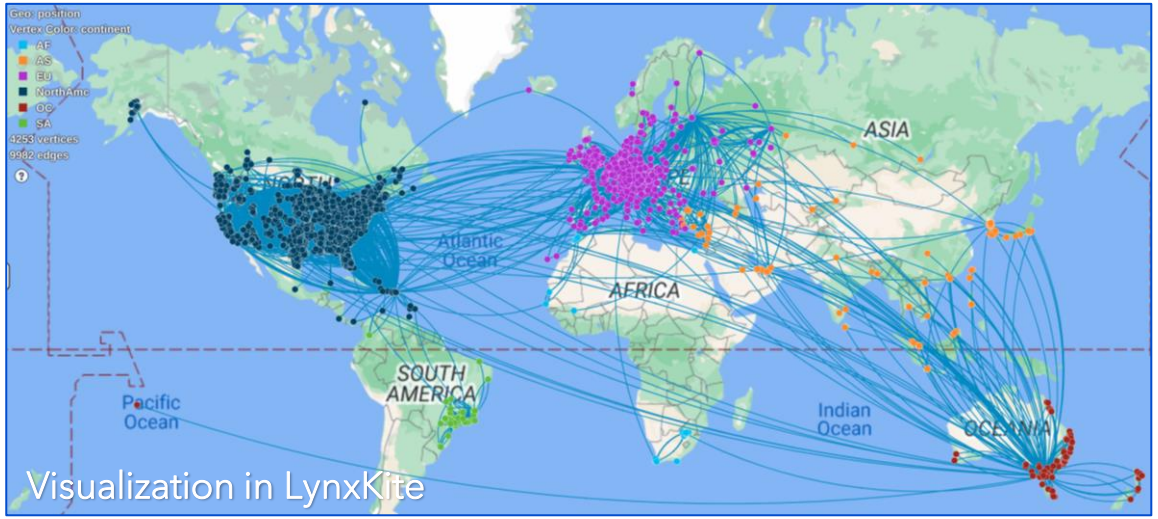
Dataset	#Nodes	#Edges	#Steps
tgbl-flight	18,143	67,169,570	1,385

Goals

Future link prediction (Given historical flight routes, predict future flight route)

Challenges

High memory and computational requirement.



Visualization in LynxKite

Introduction (9)

Objectives

Research Questions and Contributions

[1] How to efficiently analyze large-scale traffic flight prediction?

[2] Which is the most suitable TGL method? Based on which evaluation method?

[3] Can we further improve the selected TGL method?

[4] What is the generalizability of the framework and performance on full dataset?

Literature Reviews (1)

Static graph Approaches

GCN [3] (Graph Convolutional Network)

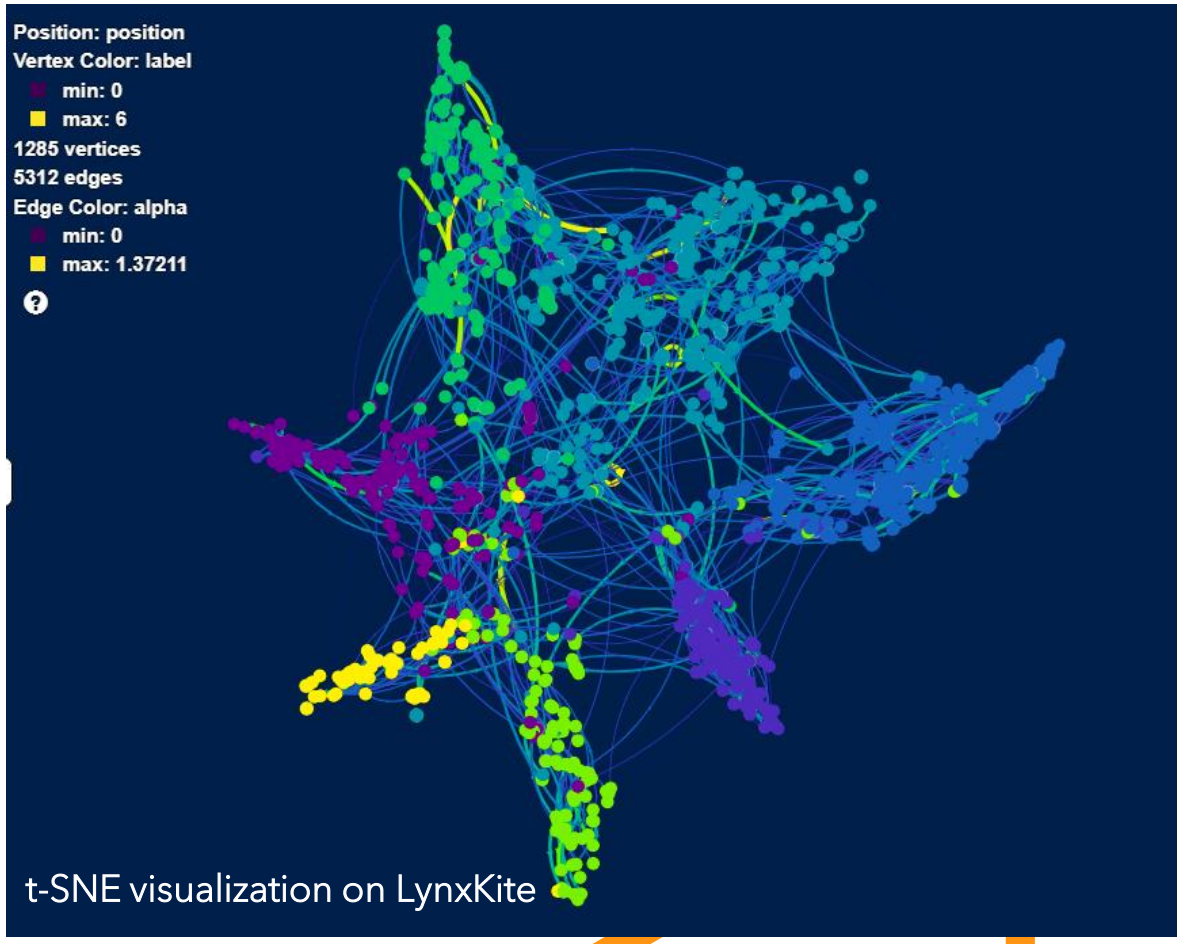
Trasductive learning method by Neighborhood aggregation normalized by node degree.

GraphSAGE [4] (Sampling and Aggregation)

Generalized GCN to Inductive setting by sampling fixed d-neighbors and aggregating by permutation invariance operator such as mean or max.

GAT [5] (Graph Attention Network)

Each node give different “importance” to their neighborhood.
An Example of Cora Graph Classification:



t-SNE visualization on LynxKite

Literature Reviews (2)

Dynamic (temporal) graph Approaches

EdgeBank[6]

Pure Memory-based evaluation methods

JODIE[7]

Dynamic projection based on user-items (bi-partite) interaction and updated by RNN.

DyRep[8]

Temporal point process-based attention mechanism

TGAT[9]

Generalize GAT to temporal graph by incorporating learnable relative time encoder derived fom Bochner's theorem.

TGN[10]

Proposed generic TGL framework consists of
(1) Message Function (2) Message Aggregator
(3) Memory Updater (4) Node Embedding
(5) Predictor

Literature Reviews (3)

Dynamic (temporal) graph Approaches

DyGFormer[11]

Turn graph learning into sequence-based learning by extracting, aligning, and patching first hop neighborhood and feeding into the Transformer. Incorporating neighbor co-occurrence to capture correlation of the neighborhoods.

TCL[12]

Use breadth first search (BFS) to extract node embedding, use two streams of attention mechanism to embed pairs of interaction node. Apply cross-attention to learn joinly between both pair of nodes.

CAWN[13]

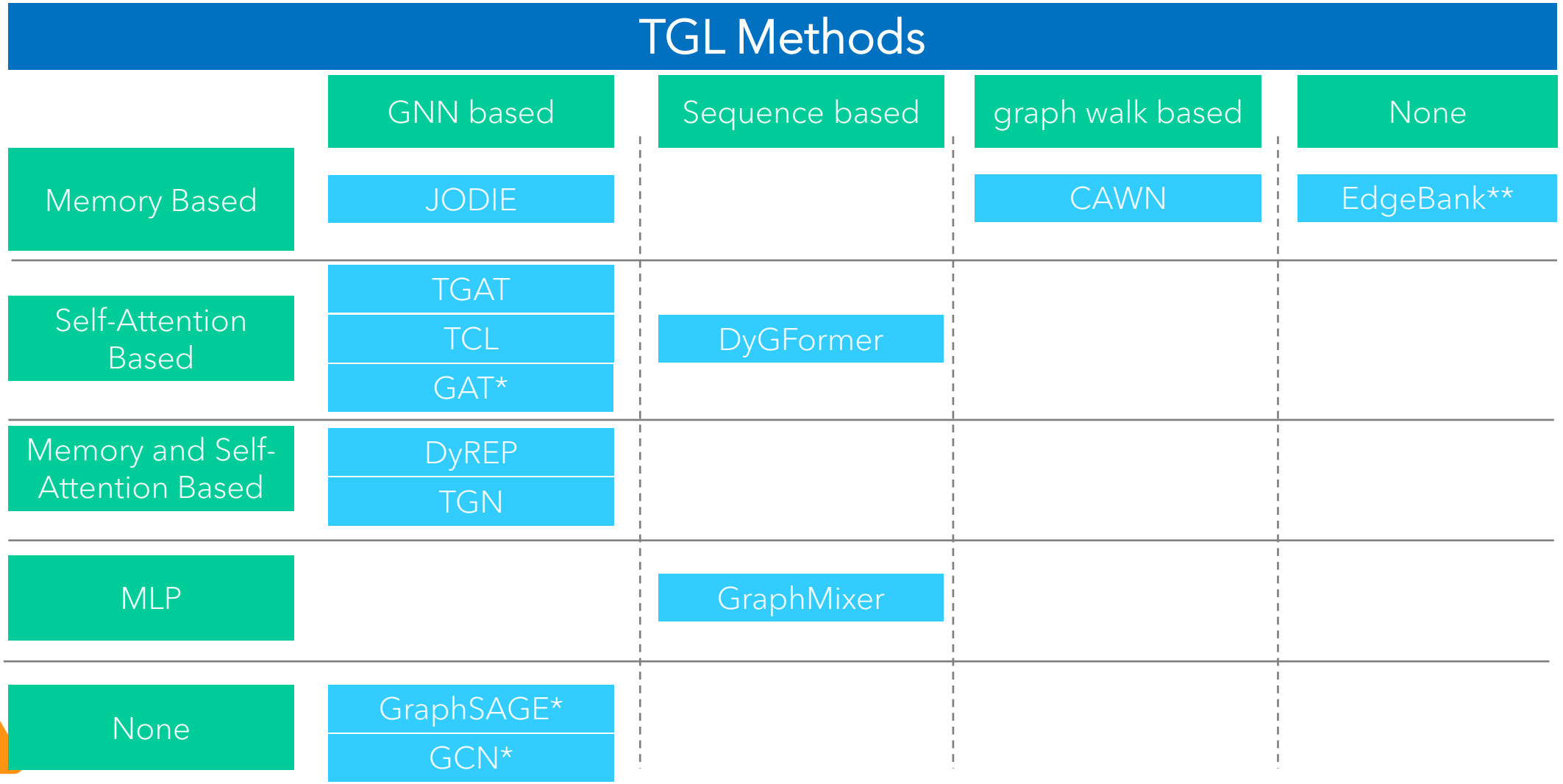
Generalized random walk into inductive setting by anonymized node identities. Using RNN to update the node embedding extracted by graph motif.

GraphMixer[14]

Propose a simple MLP-Mixer based architecture with static time encoding function

Literature Reviews (4)

TGL methods taxonomy



(*) Static graph methods. (**) Purely memory based

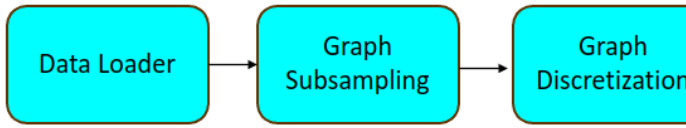
Methodology (1)

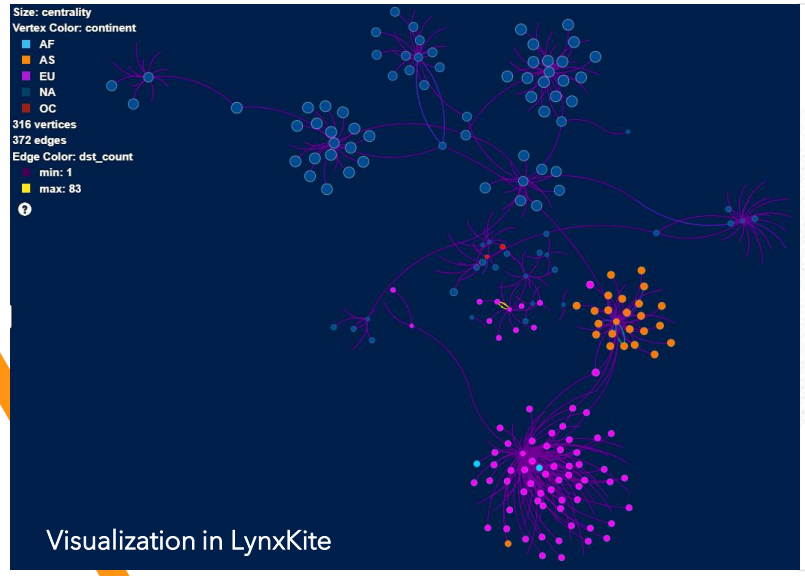
Efficient analysis approach

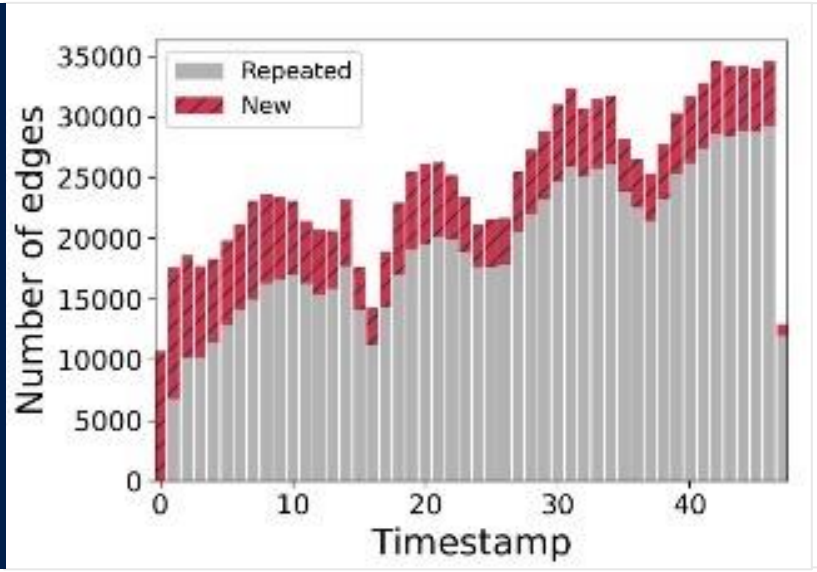
Dataset Insight

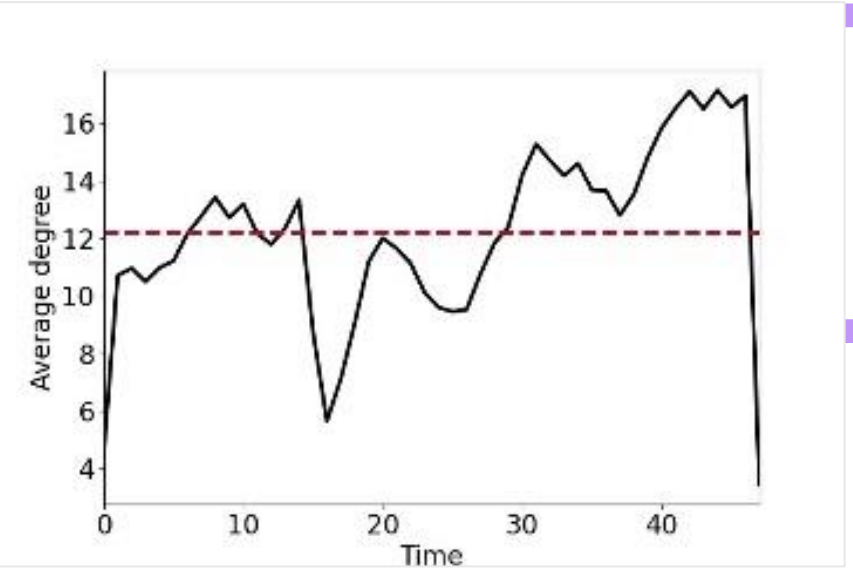
Perform graph subsampling and timestamp discretization for visualization and Statistics

	Re-occurrence	Novelty	Surprise	Average Degree
Sample Node	0.305	0.36	0.244	12.25









Methodology (2)

evaluation methods

Traditional Methods

Binary classification is considered standard

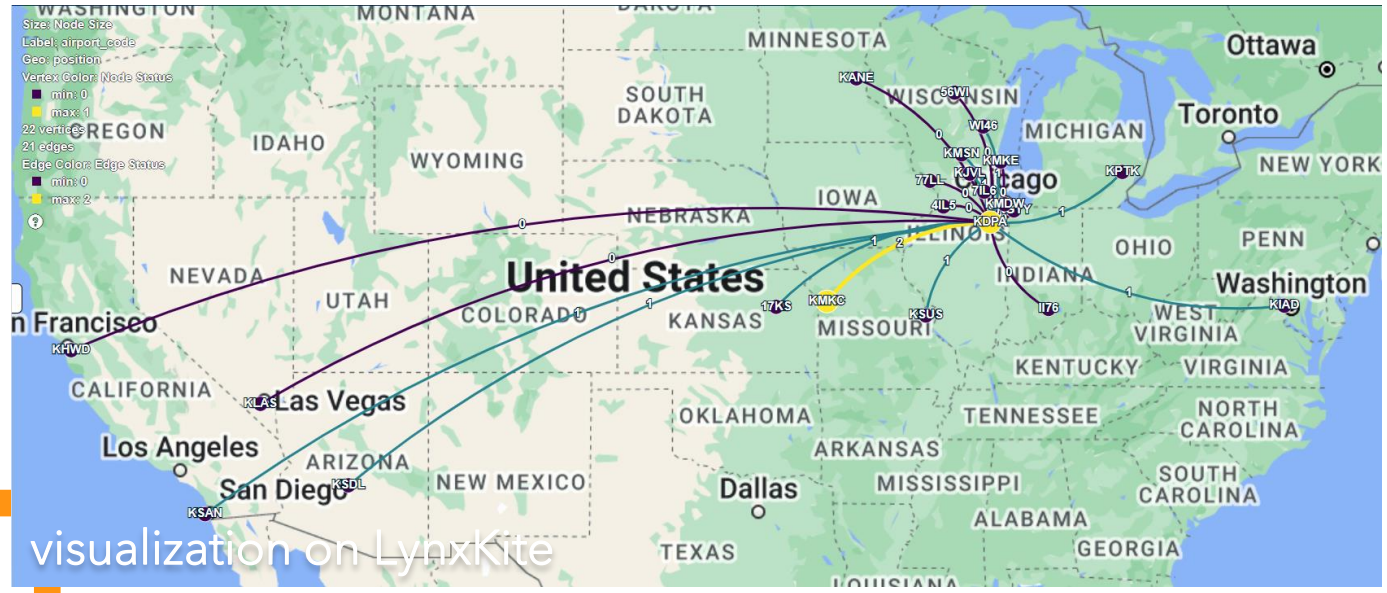
Evaluation Tasks

Link prediction using 50% historical and 50% random negative edges

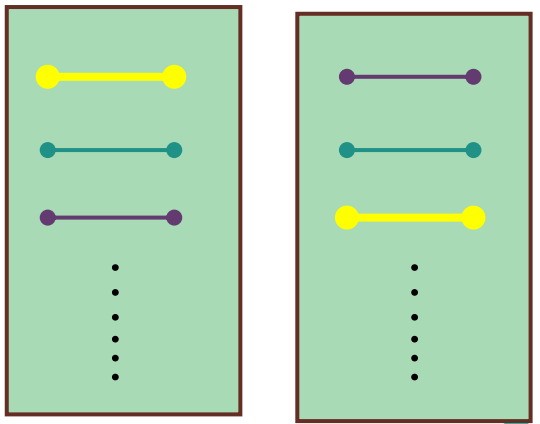
TGB Proposal

ONE-VS-MANY Rank-based evaluation with Mean Reciprocal Rank (MRR)

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$



prediction historical random



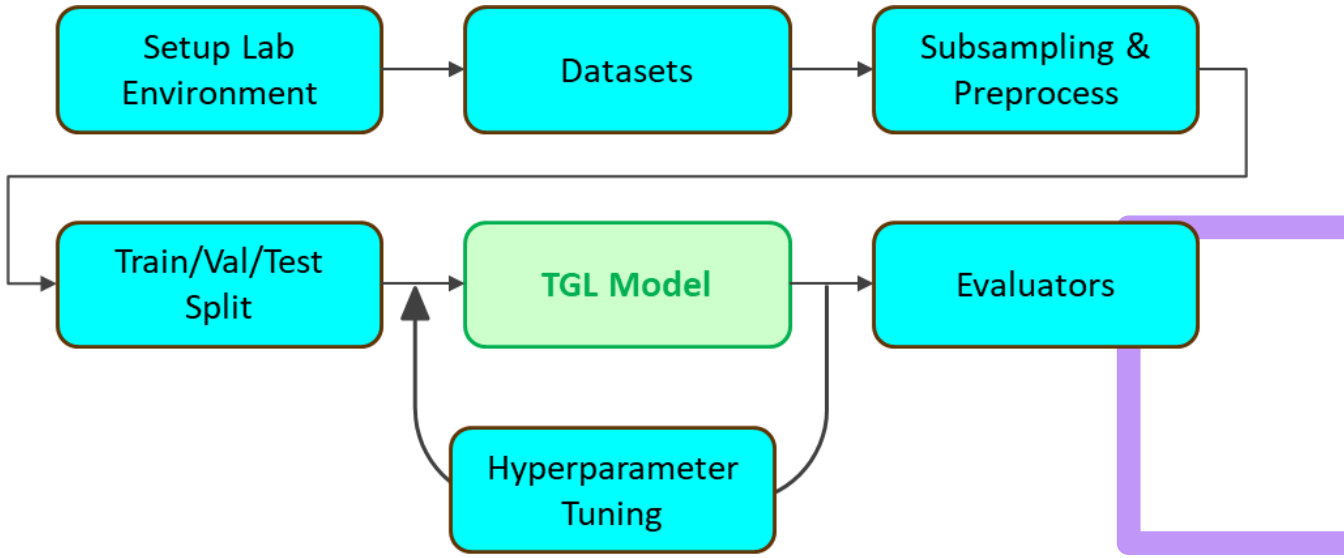
$$\text{MRR} = \frac{1}{2}(\text{RR} = 1/1 \quad + \text{RR} = 1/3)$$

Methodology (3)

Approaches and Processes

Temporal Graph Learning Pipeline

Scale-up environments for large-size datasets, split into training, validation, and test sets



Methodology (4)

Model Selection

Top performance (MRR):

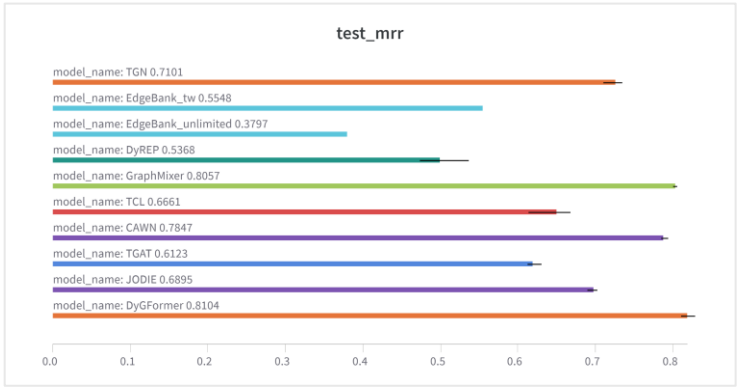
- DyGFormer: 81%
- GraphMixer: 80.57%
- CAWN: 78.47%
- TGN: 71.01%
- JODIE: 68.95%

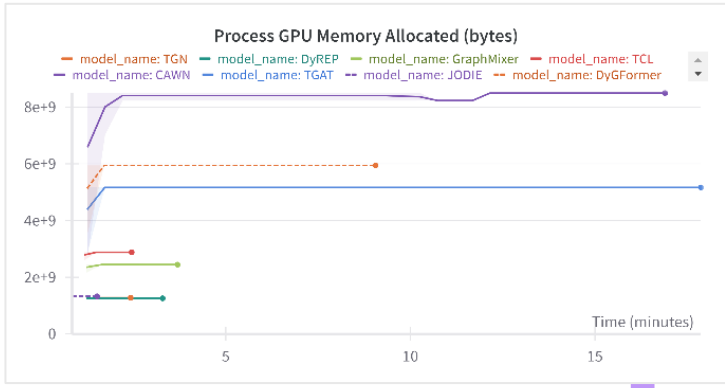
Top Efficiency (Speed):

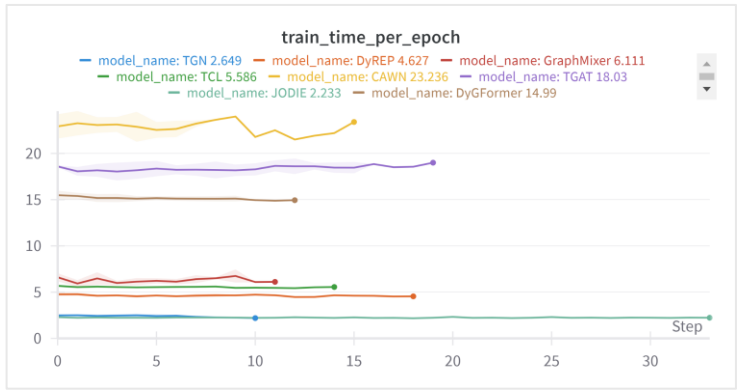
- EdgeBank (No Training)
- JODIE: 2.223s
- TGN: 2.649s
- TCL: 5.586
- GraphMixer: 6.111s

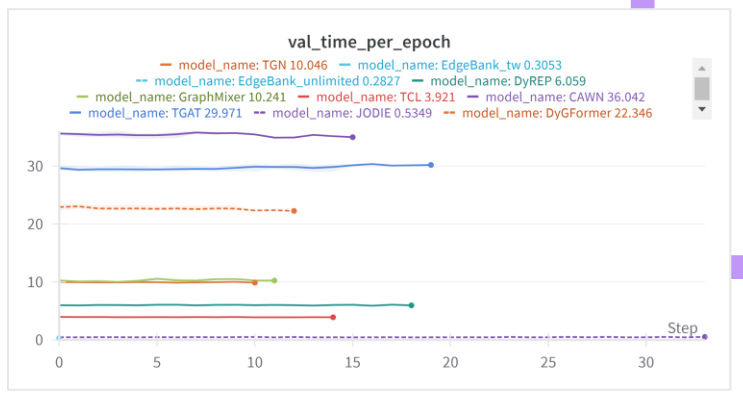
TGN is select to balance the training speed and performance

Experiment results of 10 models:









Methodology (5)

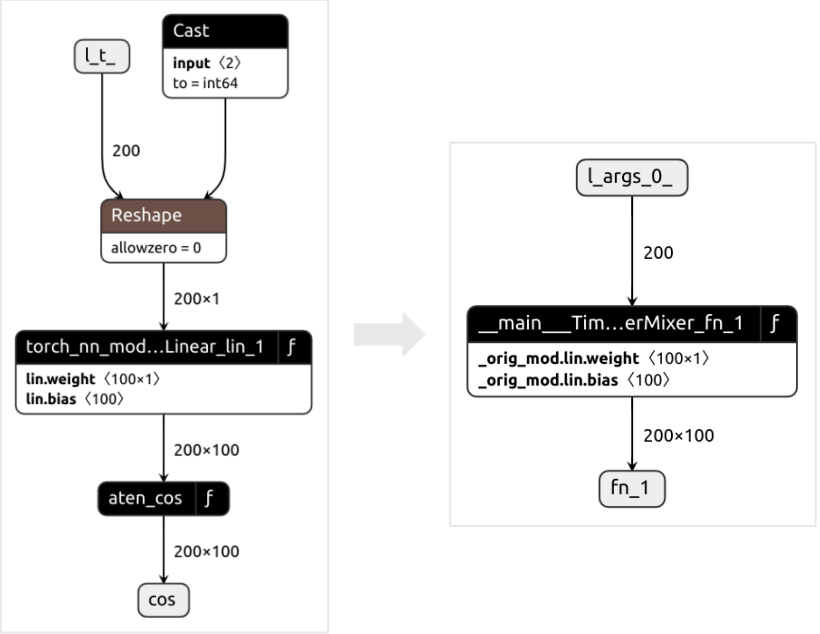
Model insight

Model Architecture

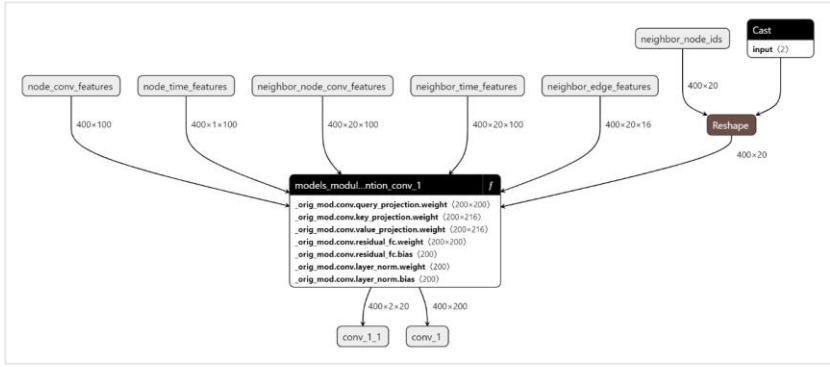
TGN consists of 3 mains modules: *Memory Updater*, *Embedding* (b), *Decoder* (c).

Propose a novel TGN with static time encoder function (a), call TGN-ST. Leverage *torch.compile* and Introduce vectorize forward pass algorithm.

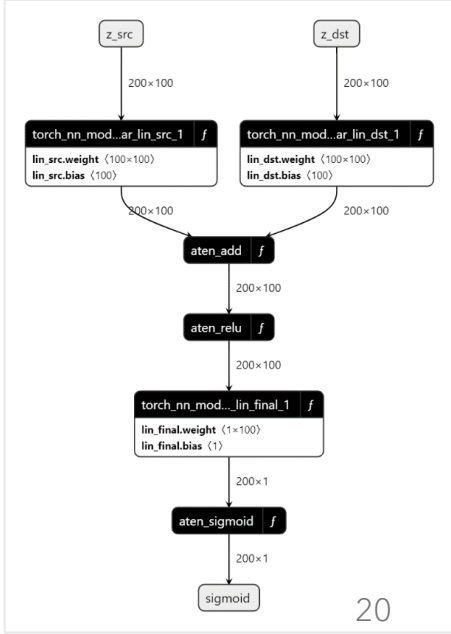
(a) Time Encoder



(b) embedding



(c) decoder



Methodology (6)

Model efficiency optimization

Algorithm 1. TGN forward pass

1	Input: $\boldsymbol{X} \in \mathbb{R}^{2\times N\times D}$; $\boldsymbol{NS} \in \mathbb{R}^{N\times S}$ ▷ $\boldsymbol{S}$ size of neighbors; $\boldsymbol{D}$ input size
2	Output: $\boldsymbol{Y} \in \mathbb{R}^{N\times 1}$; $\boldsymbol{MRR} \in \mathbb{R}$ ▷ N total number of edges
3	Initialization: mem_udater, $embedding$, $link\_predictor$
4	For each batch $\boldsymbol{X_i} \in \boldsymbol{X}$ do
5	$\boldsymbol{NS_i} \leftarrow$ Query negative dst node corresponds to each $x_i \in \boldsymbol{X_i}$ ▷ $\boldsymbol{X_i} = [\boldsymbol{X_{src_i}}, \boldsymbol{X_{dst_i}}]$
6	For each $x_i \in \boldsymbol{X_i}$ contrasts $\boldsymbol{ns_i} \in \boldsymbol{NS_i}$ do
7	$\boldsymbol{NBR_i} \leftarrow$ Query the neighbor of $\{x_i, \boldsymbol{ns_i}\}$
8	$\boldsymbol{Z_i} \leftarrow mem\_updater(\{x_i, \boldsymbol{NBR_i}\})$ ▷ implicit learnable time encoder
9	$\boldsymbol{Z_i} \leftarrow embedding(\boldsymbol{Z_i})$
10	$\boldsymbol{Y_i} \leftarrow link\_predictor(\boldsymbol{Z_{src_i}} \in \boldsymbol{Z_i}, \boldsymbol{Z_{dst_i}} \in \boldsymbol{Z_i})$ ▷ $\boldsymbol{Z_{src_i}}$ embedding of source nodes
11	$\boldsymbol{MRR_i} \leftarrow$ Compute the metrics MRR
12	End For
13	End For
14	Return $\boldsymbol{Y}$, $\boldsymbol{MRR}$

Algorithm 2. TGN-ST forward pass

1	Input: $\boldsymbol{X} \in \mathbb{R}^{2\times N\times D}$; $\boldsymbol{NS} \in \mathbb{R}^{N\times S}$ ▷ $\boldsymbol{S}$ size of neighbors; $\boldsymbol{D}$ input size
2	Output: $\boldsymbol{Y} \in \mathbb{R}^{N\times 1}$; $\boldsymbol{MRR} \in \mathbb{R}$ ▷ N total number of edges
3	Initialization: $mem\_updater$, $torch.compile(\{embedding, link\_predictor\})$
4	For each batch $\boldsymbol{X_i} \in \boldsymbol{X}$ do
5	$\boldsymbol{NS_i} \leftarrow$ Query negative dst node corresponds to $\boldsymbol{X_i}$ ▷ $\boldsymbol{X_i} = [\boldsymbol{X_{src_i}}, \boldsymbol{X_{dst_i}}]$
6	$\boldsymbol{NBR_i} \leftarrow$ Query the neighbor of $\{\boldsymbol{X_i}, \boldsymbol{NS_i}\}$
7	$\boldsymbol{Z_i} \leftarrow mem\_updater(\boldsymbol{X_i}, \boldsymbol{NBR_i})$ ▷ implicit static time encoder
8	$\boldsymbol{Z_i} \leftarrow embedding(\boldsymbol{Z_i})$
9	$\boldsymbol{Y_i} \leftarrow link\_predictor([\boldsymbol{Z_{src_i}} \in \boldsymbol{Z_i}, \boldsymbol{Z_{dst_i}} \in \boldsymbol{Z_i}])$ ▷ $\boldsymbol{Z_{src_i}}$ embedding of source nodes
10	For each $x_i \in \boldsymbol{X_i}$ contrasts $\boldsymbol{ns_i} \in \boldsymbol{NS_i}$ do
11	$\boldsymbol{MRR_i} \leftarrow$ Compute the metrics MRR
12	End For
13	End For
14	Return $\boldsymbol{Y}$, $\boldsymbol{MRR}$

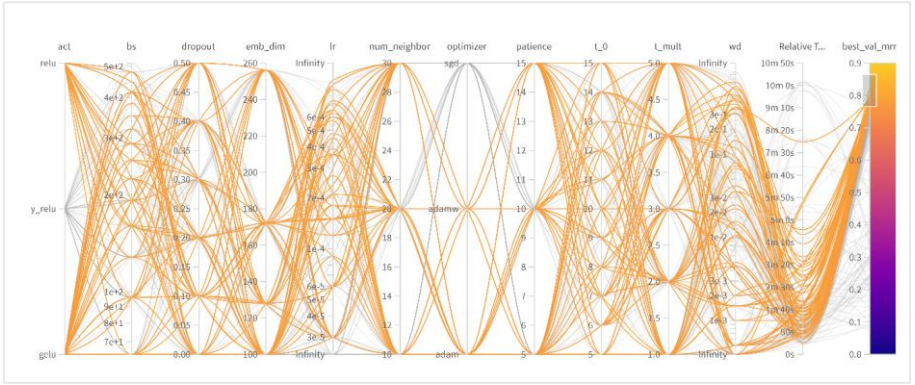
Methodology (7)

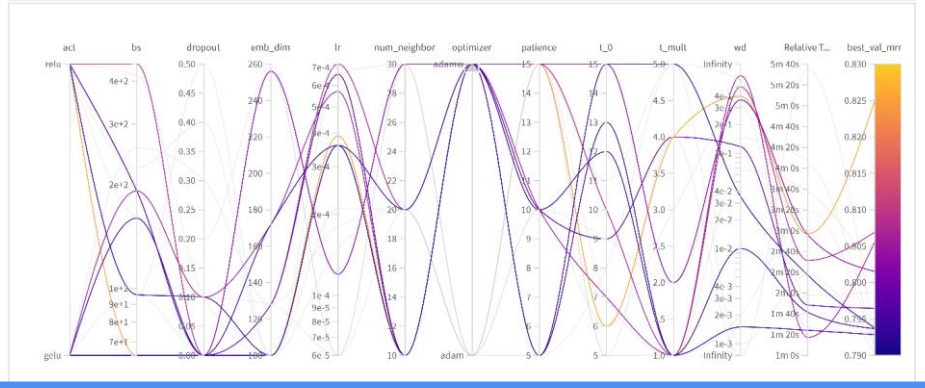
Model performance optimization

Hyperparameter Tuning

Perform 200 random search experiments:

- -SGD is the worst optimizer
- -Leaky_relu activation function should be avoid
- -AdamW with cosine annealing is the best training strategy.
- -Batch size & patience number directly impact the training speed.





	Act	bs	dropout	emb_dim	lr	num_neighbor	optimizer	patience	t_0	t_{mult}	wd
distribution	rand_sel	q_log_uniform_values	int_uniform	rand_sel	q_log_uniform_values	rand_sel	rand_sel	rand_sel	int_uniform	int_uniform	q_log_uniform_values
range	[relu, leaky_relu, gelu]	[32,512]	[0,0.5]	[100,172, 256]	[0.00001, 0.0001]	[10,20,30]	[sgd, adam, adamw]	[5,10,15]	[5,15]	[1,5]	[0.0001, 1]
step	-	32	0.1	-	0.00003	-	-	-	1	1	0.0005
final selection	ReLU	288	0.1	100	0.00072	10	AdamW	5	14	4	0.01292

Results and Discussions

Findings

Sample Dataset Result

-TGN-ST gains 7.25 improvement
-25% faster training time
-4x Validation time

Full Dataset Result

- TGN-ST sets new state-of-the-art result with 72.49% MRR
- 15% faster training time
- 5x Validation time

Implication

Consistent improvement on both dataset provide evident for the generalizability of the pipeline and TGN-ST forward pass algorithm.

	Model and Gains	Test MRR (%)	Training Time per Epoch (s)	Validation Time per Epoch (s)
100 Sample Nodes	Vanilla TGN	72.94	2.3743	9.8546
	TGN-ST	80.19	1.7763	2.2687
	Gain	7.25	0.598 (25%)	7.586 (4.34x)
Full Dataset	Vanilla TGN	70.50 (*)	2800 (**)	9485 (**)
	TGN-ST	72.49	2366	1887
	Gain	1.99	434 (15%)	7598 (5.02x)

(*) Result obtained from TGB leaderboard. (https://tgb.complexdatalab.com/docs/leader_linkprop/)
(**) Result obtained by running & averaging first few epochs

Conclusions and Future Directions

Summary and Future Work

Scalable Framework

Established scalable framework, performed graph subsampling, visualization, hyperparameter tuning

Model Improvement

Proposed TGN-ST model with new state-of-the-art results and with significant training speed and evaluation speed improvement.

Future Direction

[1] Explore different graph sampling strategy such as random walk sampling

[2] Apply framework to other TGB datasets

[3] Model parallelism and scale further to multi-GPU or multi-node cluster

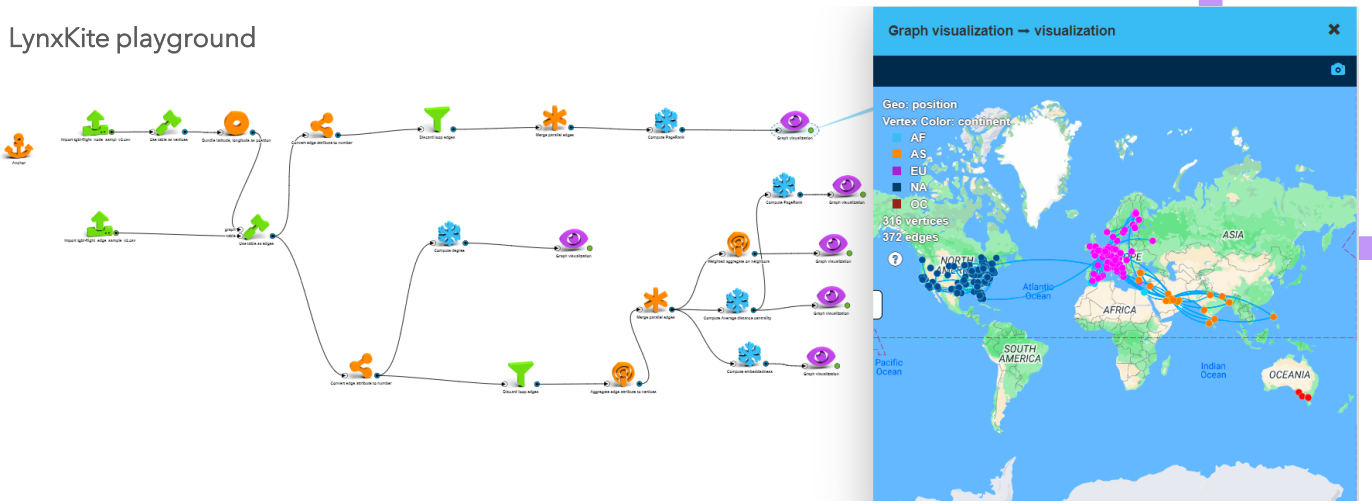
[4] Apply more complex neighborhood aggregation method and long-range dependency memory updater such as Selective State-Space Model (S6) or xLSTM

Publication

Hang H., Sokhey P., Gabor B. Sothea H., and Pheak N. Temporal Graph Learning with Application to Large-Scale Traffic Flight Prediction. In Techno-SRJ (under review), 2024.

Acknowledgement

- Dr. Gabor's guides on graph visualization on Lynxkite
- Lynx Analytics' funded Lynx azure cluster



References

[1] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. IEEE Conference on Computer Vision and Pattern Recognition.

[2] Pan, Z. W., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A Comprehensive Survey on Graph Neural Networks. arXiv:1901.00596.

[3] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. International Conference on Learning Representations (ICLR).

[4] Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in Neural Information Processing Systems (NeurIPS).

[5] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. International Conference on Learning Representations (ICLR).

[6] Poursafaei, F., Huang, S., Pelrine, K., & Rabbany, R. (2022). Towards Better Evaluation for Dynamic Link Prediction. Neural Information Processing Systems (NeurIPS).

[7] Kumar, S., Zhang, X., & Leskovec, J. (2019). Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. ACM SIGKDD.

[8] Trivedi, R., Farajtabar, M., Biswal, P., & Zha, H. (2019). Dyrep: Learning representations over dynamic graphs. International conference on learning representations.

[9] Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., & Achan, K. (2020). Inductive Representation Learning on Temporal Graphs. International Conference on Learning Representations.

[10] Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2021). Temporal Graph Networks for Deep Learning on Dynamic Graphs. ICLR.

[11] Yu, L., Sun, L., Du, B., & Lv, W. (2023). Towards Better Dynamic Graph Learning: New Architecture and Unified Library. Conference on Neural Information Processing Systems (NIPS).

[12] Wang, L., Chang, X., Li, S., Chu, Y., Li, H., Zhang, W., . . . Yang, H. (2018). TCL: Transformer-based Dynamic Graph Modelling via Contrastive Learning. arXiv preprint arXiv:2105.07944.

[13] Wang, Y., Chang, Y.-Y., Liu, Y., Leskovec, J., & Li, P. (2021). Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. International Conference on Learning Representations (ICLR).

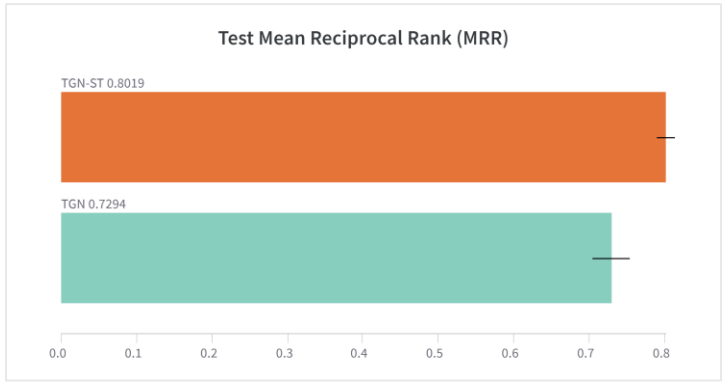
[14] Cong, W., Zhang, S., Kang, J., Yuan, B., Wu, H., Zhou, X., . . . Mahdavi, M. (2023). Do We Really Need Complicated Model Architectures For Temporal Networks? International Conference on Learning Representations (ICLR).

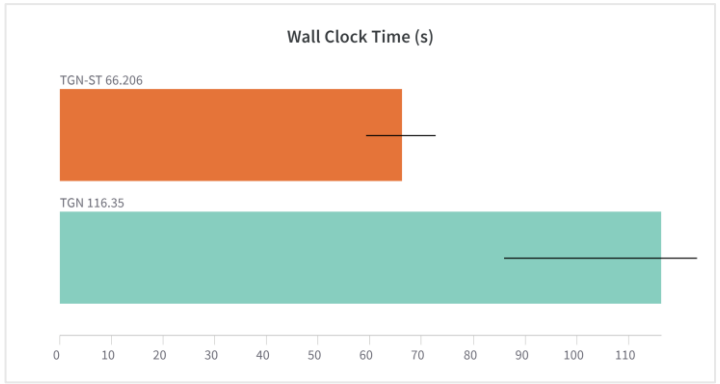
[15] Huang, S., Danovitch, F. P., Fey, M., Hu, W., Rossi, E., Leskovec, J., . . . Rabbany, R. (2023). Temporal Graph Benchmark for Machine Learning on Temporal Graphs. Advances in Neural Information Processing Systems.

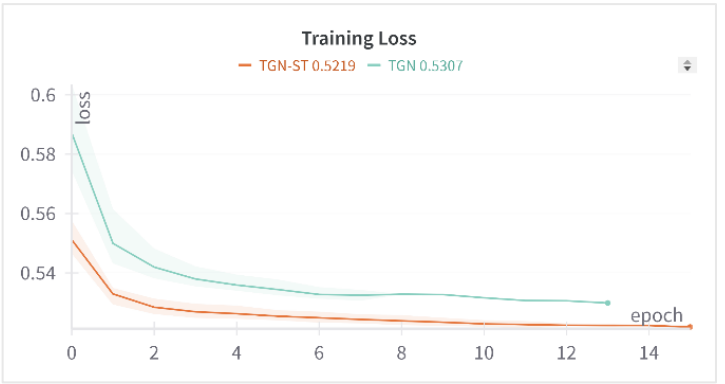
Thank you

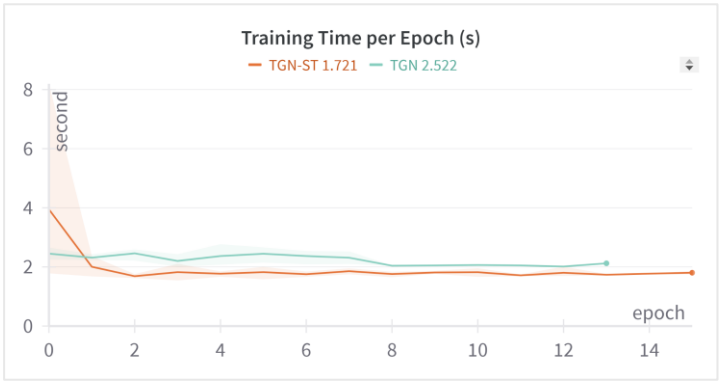
Q & A

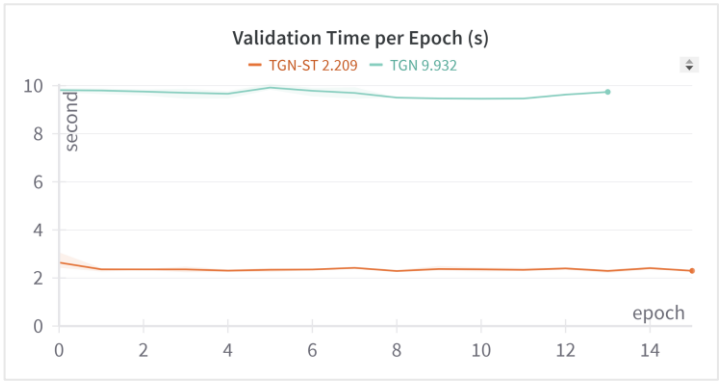
Appendix 1. TGN-ST vs TGN Benchmarking

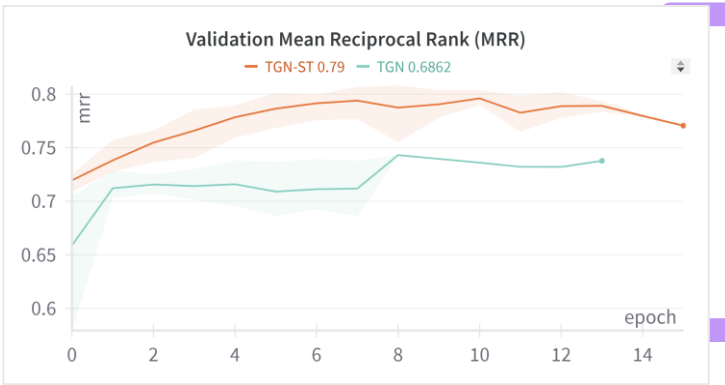




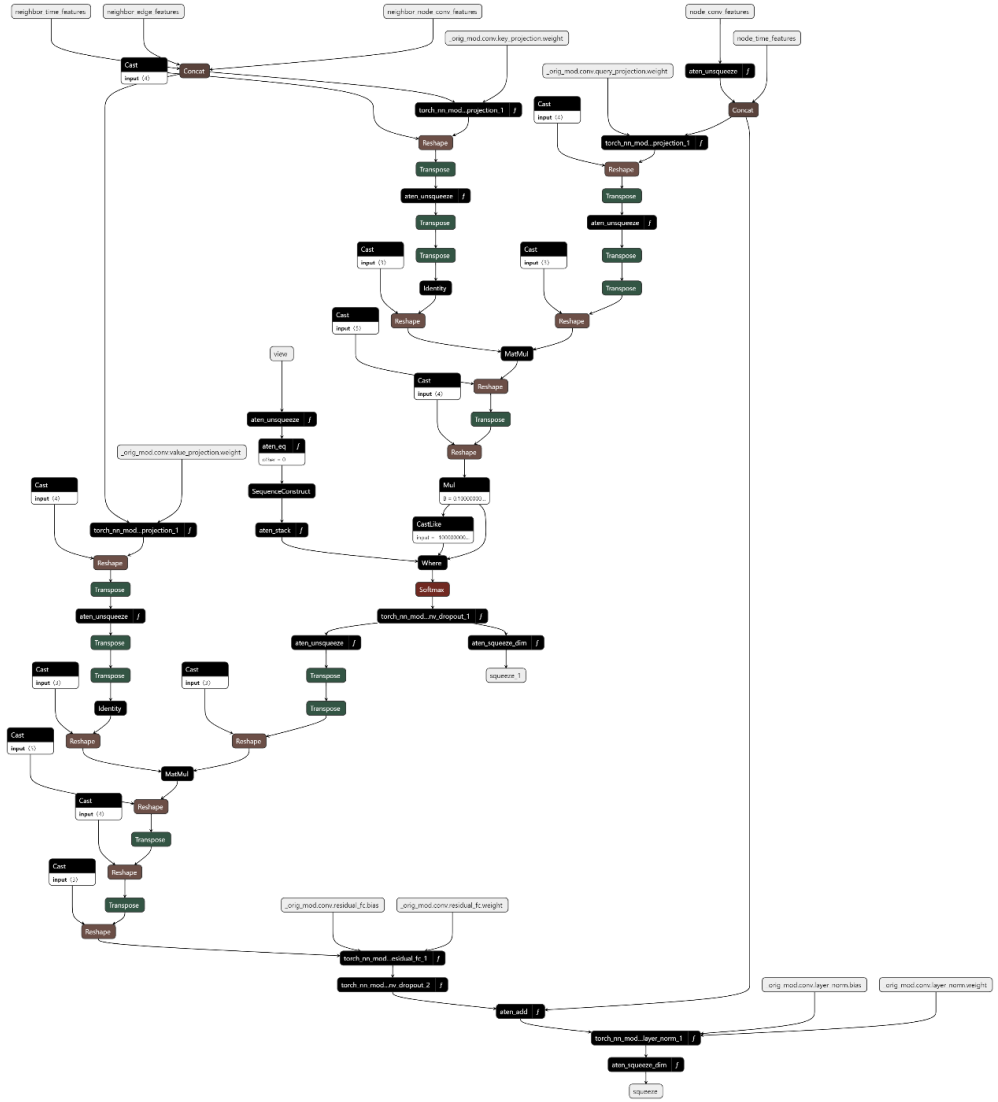








Appendix 2. TGN embedding module's FX Graph



Appendix 3. Experimental Setups

- Hardware and Software Requirements:
  - Main computing machine
    - Operating System: Ubuntu 22.04
    - Python Version: 3.10 or higher
    - CPU: Intel Core i7 14700k @3.4GHz with 20 cores (8 P-cores + 12 E-cores).
    - GPU: NVIDIA RTX 4080 Super (320 tensor cores) with 16GB memory.
  - Additional computing machine (Lynx Azure cluster provided by Lynx Analytics)
    - Operating System: Ubuntu 22.04
    - Python Version: 3.10 or higher
    - Cluster: Standard NC4as T4 v3 (4 vcpus, 28 GiB memory)
    - CPU: AMD EPYC 7V12(Rome) CPUs
    - GPU: Nvidia Tesla T4 with 16GB memory

- Main Software Libraries:
  - PyTorch (version 2.0 or higher)
  - PyTorch Geometric (version 2.5 or higher)
  - Wandb (https://wandb.ai/)

- Graph visualization and FX graph capture tools:
  - Lynxkite by Lynx Analytics (https://try.lynxkite.com/)
  - Onnx export (https://onnx.ai/)