Fundamental Problems in Graph ML: Optimization Generalization, Privacy, and Model design

Weilin Cong

Penn State University

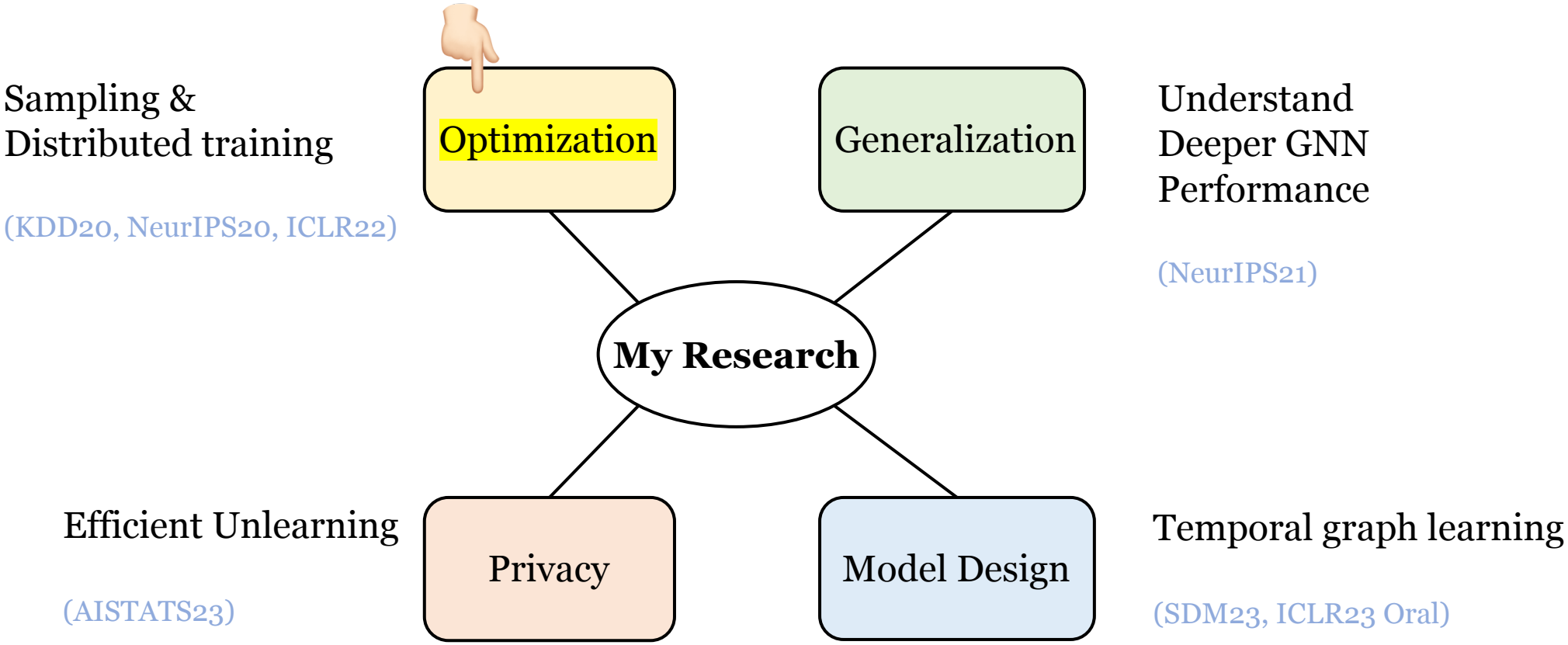
PennState

Outline

- Overview on my PhD research study
- Briefly summarization of my research views
- Deep delve into most recent publication at ICLR23
- Future directions

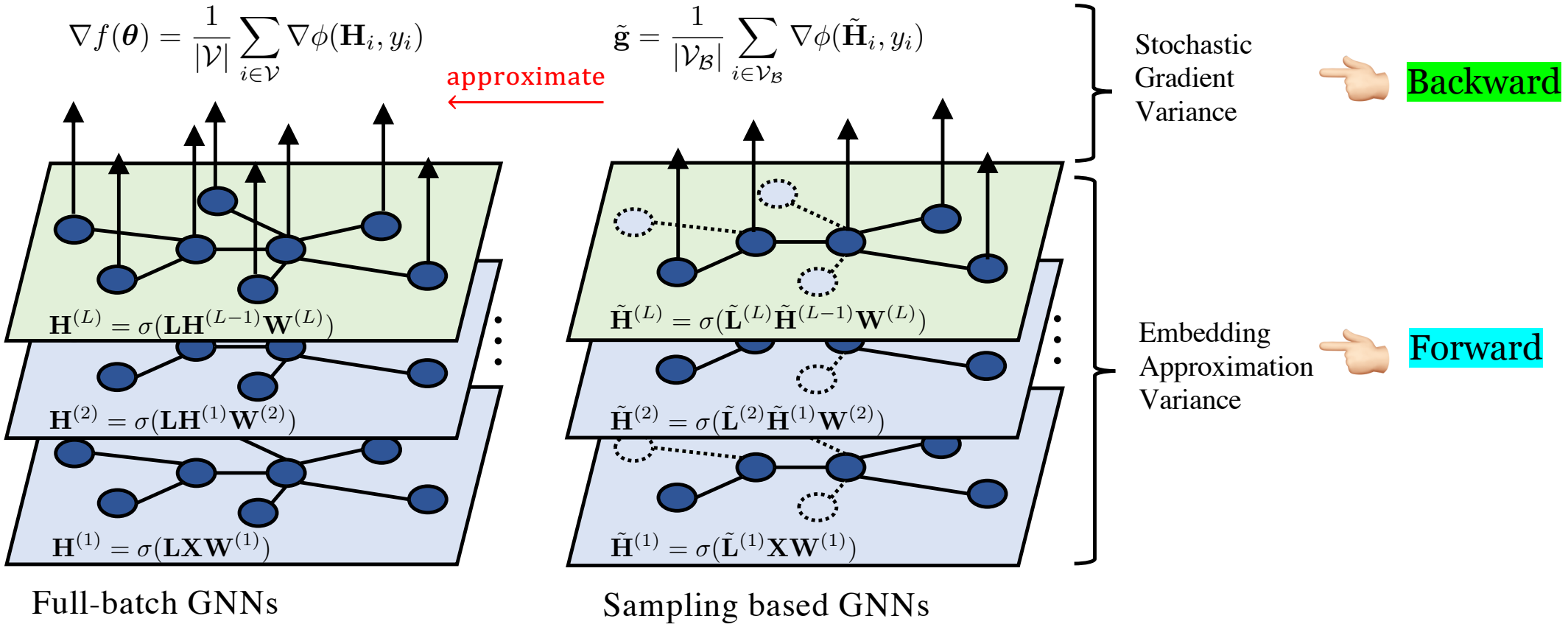
Overview on research

- During my PhD study, my research focuses on fundamental machine learning problems on graph structured data



Optimization, Generalization, Privacy, Model design

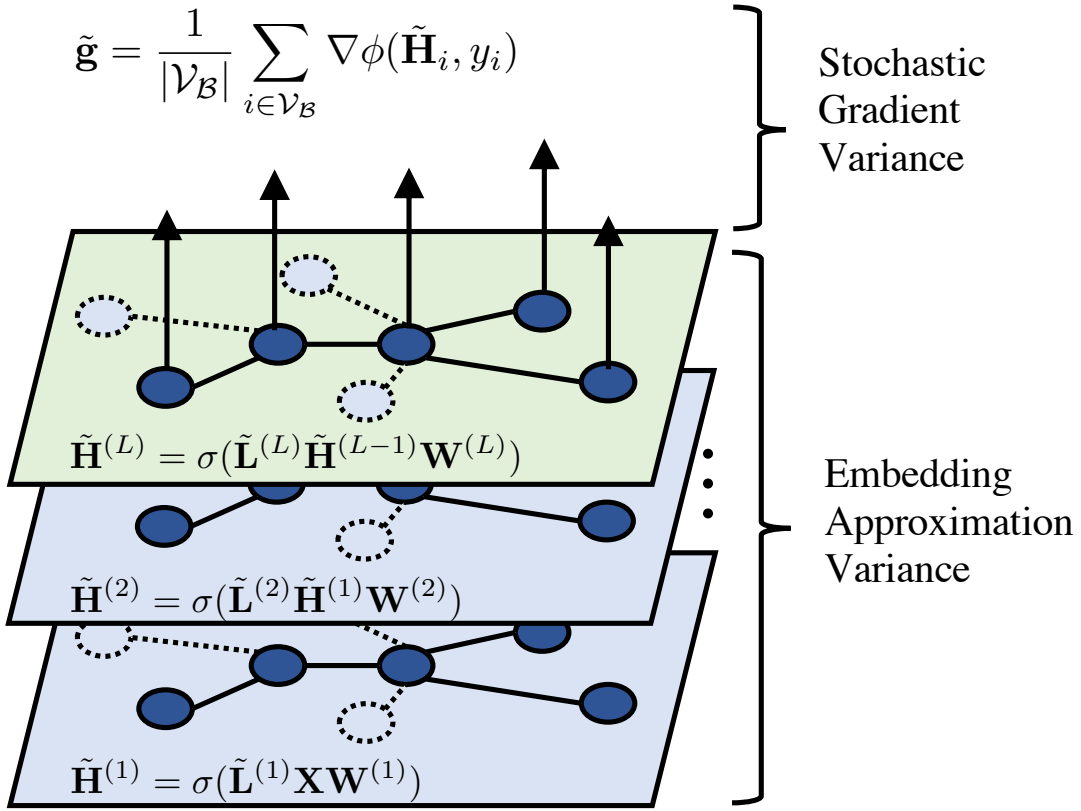
- (Theory) Due to the composite structure of empirical risks, the stochastic gradient is a biased estimation of full-batch gradient and can be decomposed into two types of variances. We must mitigate both types of variance to obtain faster convergence rate.



Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi.
Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks (KDD20)

Optimization, Generalization, Privacy, Model design

- (Algorithm) A decoupled variance reduction strategy that employs the dynamic information during optimization to sample nodes



Sampling based GNNs

👈 Selecting mini-batch nodes with probability proportional to the gradient norm computed on each node

$$\min_{\boldsymbol{p}} \quad \mathbb{E}[\|\mathbf{g} - \mathbb{E}[\mathbf{g}]\|_2^2]$$

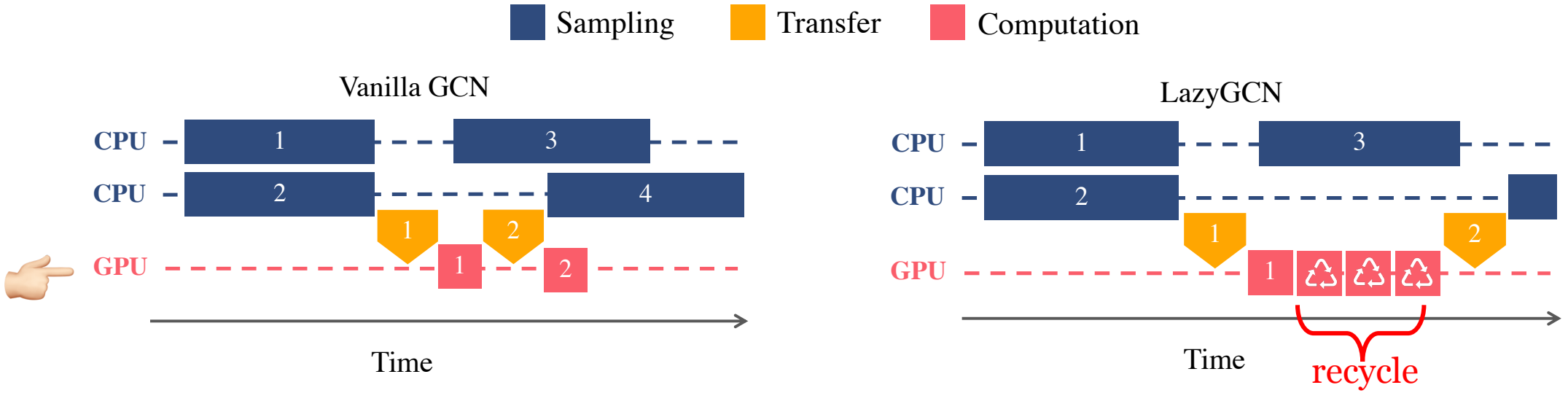
$$\text{subject to } \sum_{i=1}^{N} p_i = |\mathcal{B}|, p_i \in (0, 1]$$

👈 Using historical node hidden embeddings

Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi.
Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks (KDD20)

Optimization, Generalization, Privacy, Model design

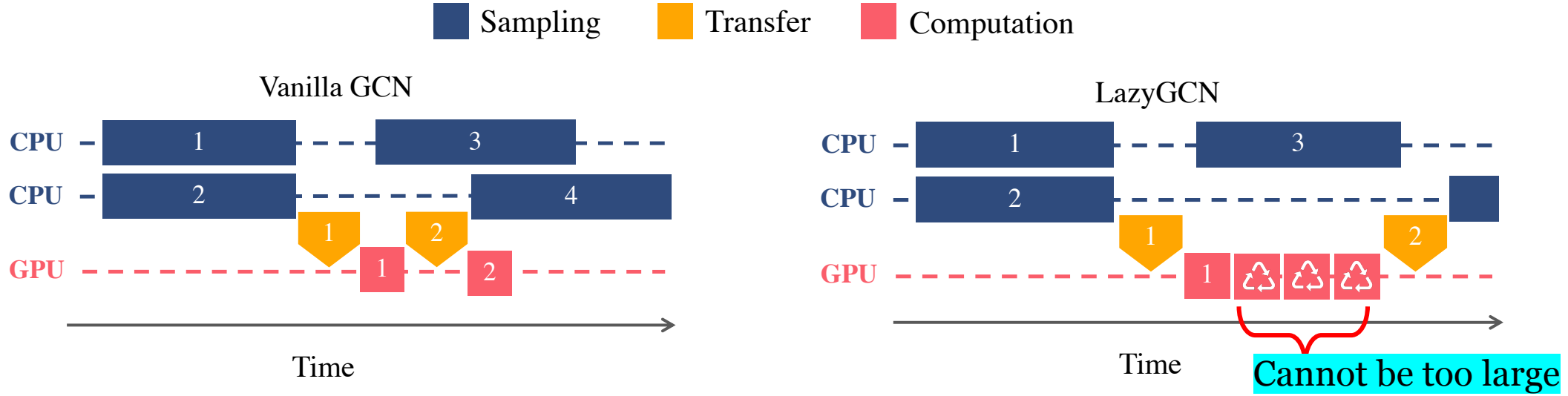
- Due to bandwidth and memory bottlenecks, sampling-based GNN training has high overhead in "pre-processing" and "loading new samples"
- (Left figure) the fraction of **computation time** (on GPU) is small compared to the **sampling** and **data-transfer** time (on CPU).
- (Algorithm) Perform node sampling periodically and recycling the sampled nodes to mitigate data preparation overhead, as shown in the right figure.



Morteza Ramezani*, **Weilin Cong***, Mehrdad Mahdavi, Anand Sivasubramaniam, Mahmut Kandemir.
GCN meets GPU: Decoupling "When to Sample" from "How to Sample" (*NeurIPS20*)

Optimization, Generalization, Privacy, Model design

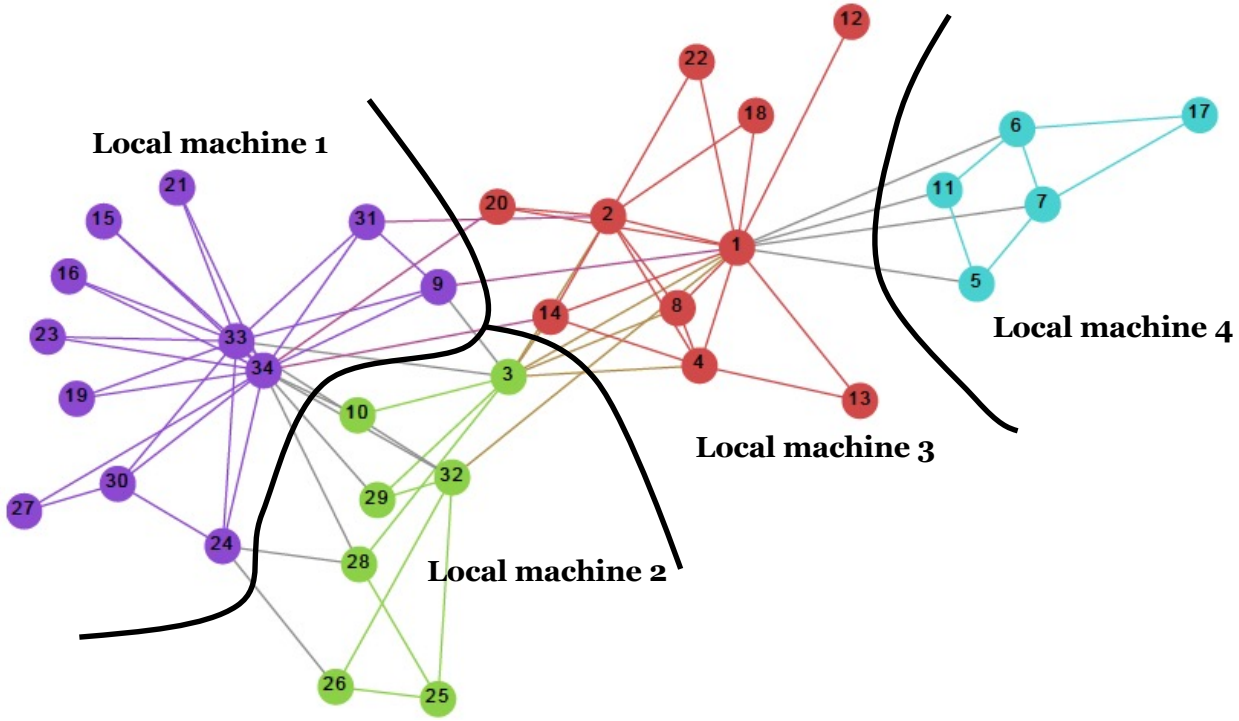
- (Theory) We show that under mild conditions on the gap between two sampling periods, by reducing the variance of inner layer sampling, the same convergence rate as the underlying sampling method can be achieved.
- "Reducing the variance of inner layer sampling" refer to fixing the inner layer nodes while recycling to those sampled at the beginning of recycling stage, but only sample the last layer nodes

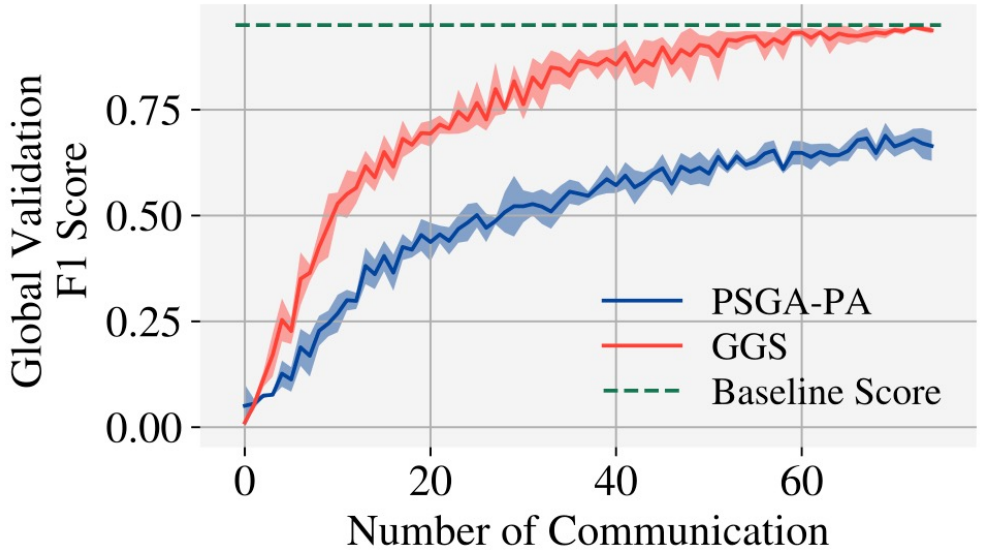


Morteza Ramezani*, **Weilin Cong***, Mehrdad Mahdavi, Anand Sivasubramaniam, Mahmut Kandemir.
GCN meets GPU: Decoupling "When to Sample" from "How to Sample" (*NeurIPS20*)

Optimization, Generalization, Privacy, Model design

- Partitioning the original graph into multiple subgraphs, each subgraph is trained on single local machine with periodic parameter averaging. However, graph partitioning will lead to subgraphs with edges spanning subgraphs.
- Consider them edge iterations? ☹
- Ignoring these spanning subgraph edges? ☹



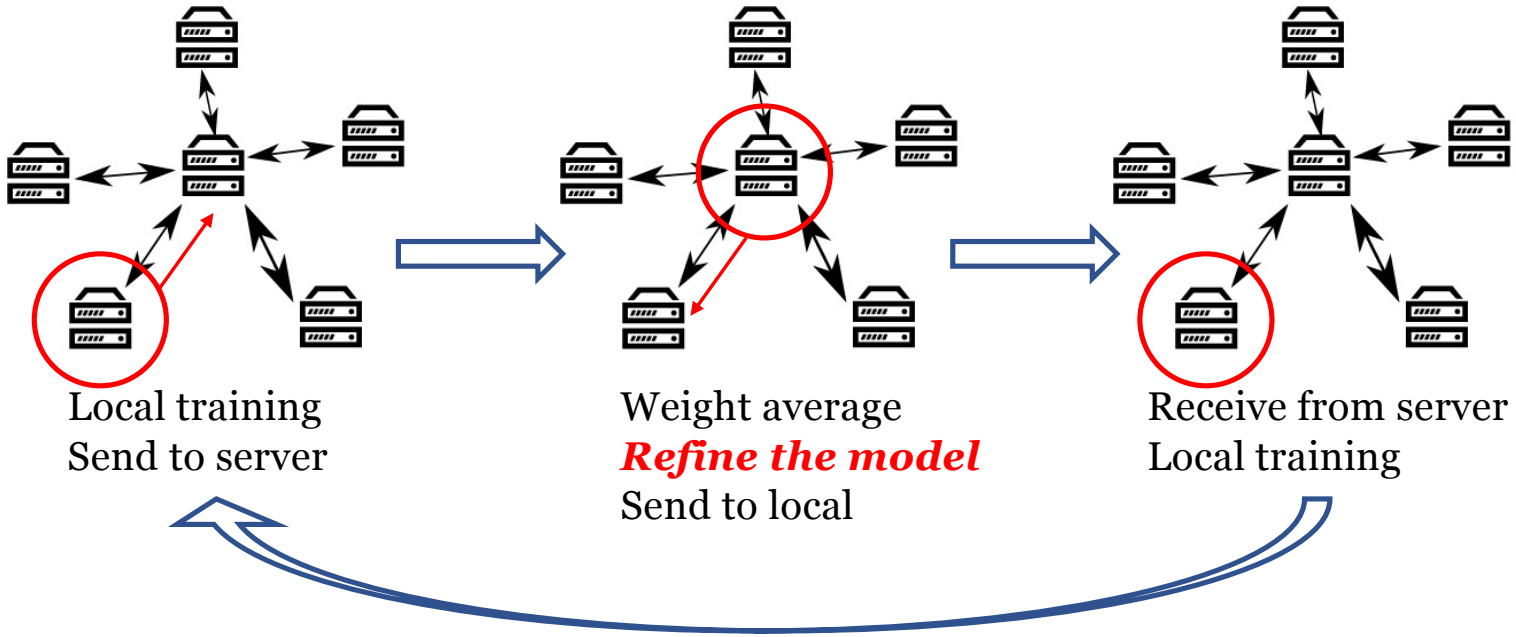


Green: single machine
Red: considering all spanning edges
Blue: ignore all spanning edges

Morteza Ramezani*, **Weilin Cong***, Mehrdad Mahdavi, Mahmut Kandemir, Anand Sivasubramaniam.
Learn Locally, Correct Globally: A Distributed Algorithm for Training Graph Neural Networks. (*ICLR22*)

Optimization, Generalization, Privacy, Model design

- (Algorithm) We propose to locally train the model on each local machine for several epochs, then perform server correction (i.e., refine the model) to mitigate the gradient bias issue.

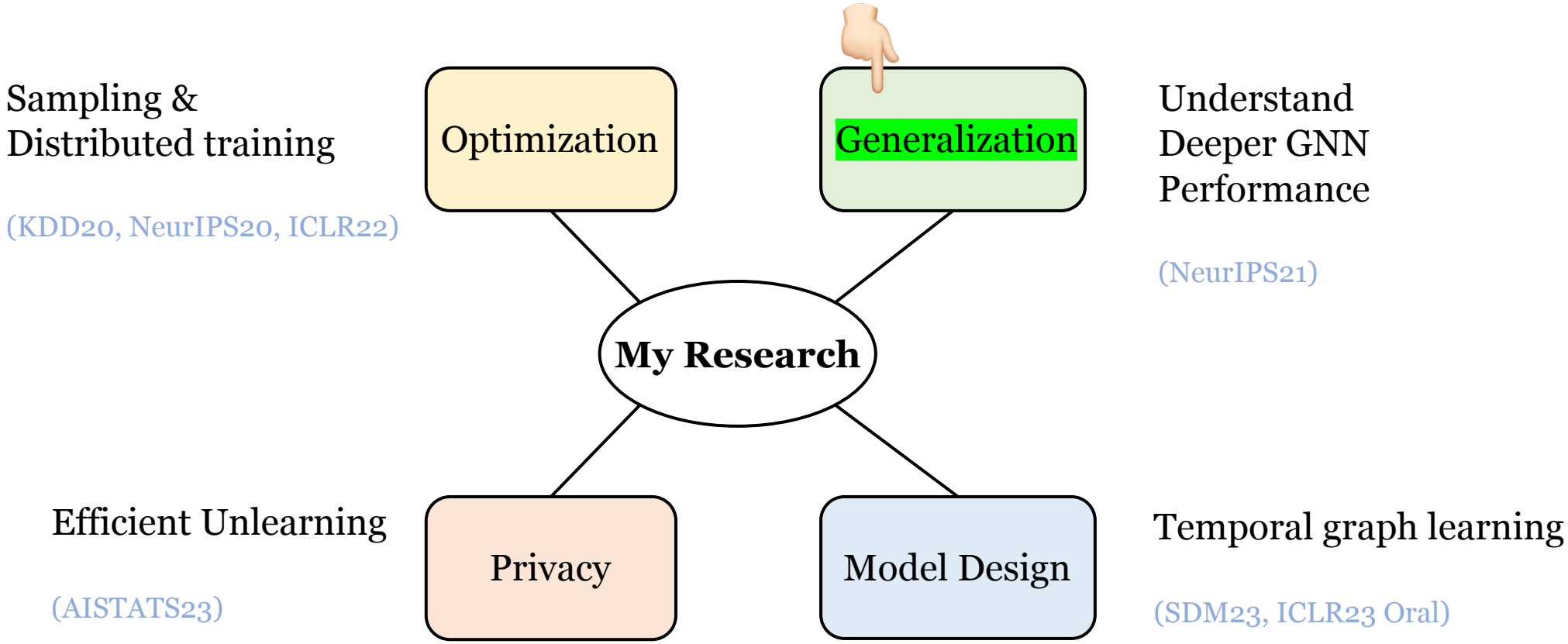


- (Theory) Ignoring the edges that spanning subgraphs will suffer from an irreducible residual error. This error can be eliminated by server correction.

Morteza Ramezani*, **Weilin Cong***, Mehrdad Mahdavi, Mahmut Kandemir, Anand Sivasubramaniam. Learn Locally, Correct Globally: A Distributed Algorithm for Training Graph Neural Networks. (*ICLR22*)

Overview on research

- During my PhD study, my research focuses on fundamental machine learning problems on graph structured data



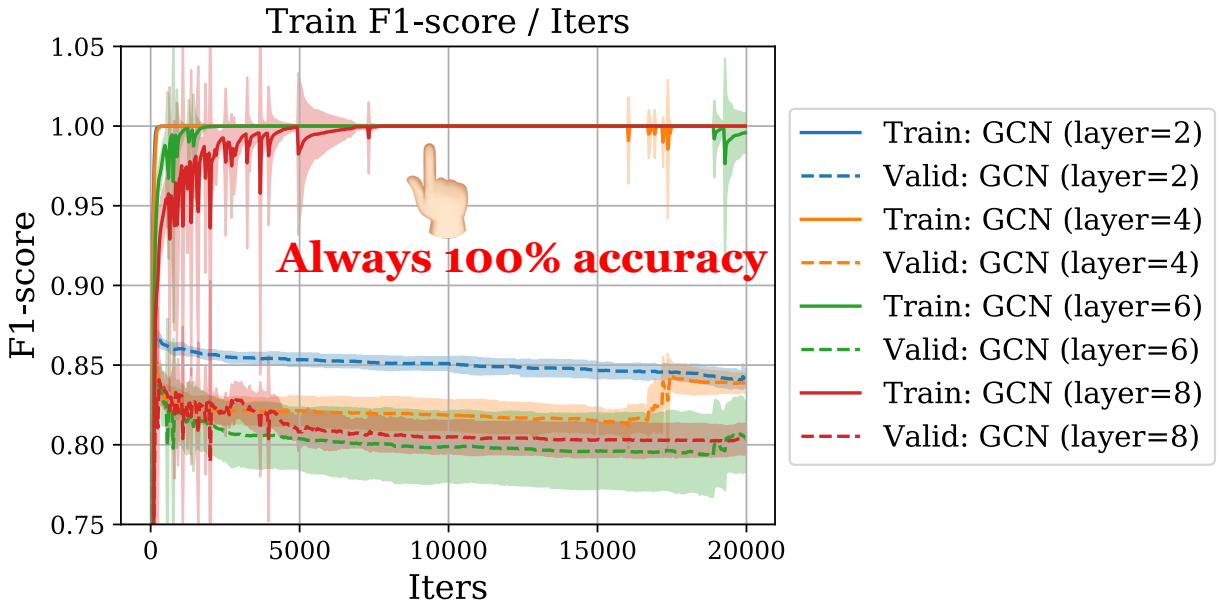
Optimization, **Generalization**, Privacy, Model design

- Performance degradation in deeper GNN is commonly explained by “over-smoothing ”。
- **Over-smoothing:** The node representation becomes indistinguishable after too many graph convolutional layers. As a result, the classifier has difficulty assigning the correct label for each node if over-smoothing happens.
- However, we argue that ”over-smoothing” not necessarily happen in practice.

Use DGL’s official implementations

Remove Dropout

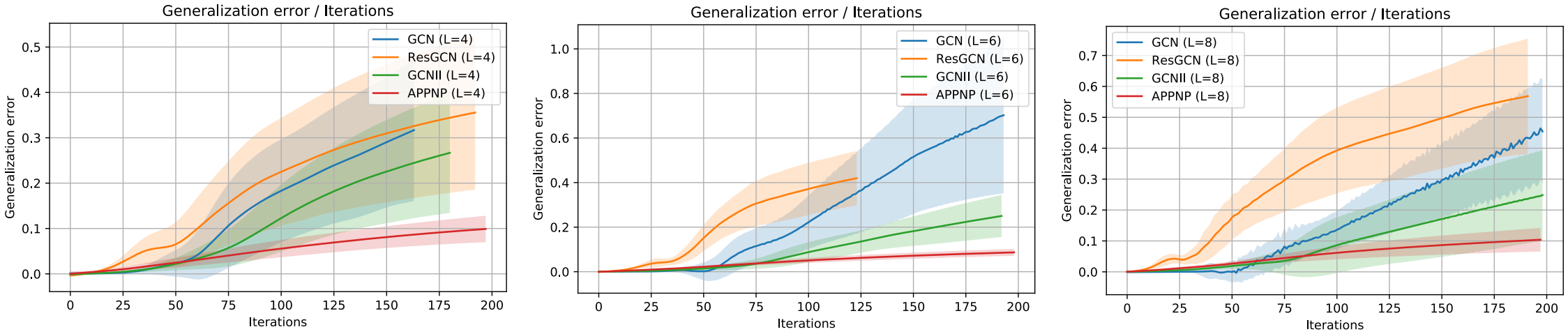
Remove weight decay (regularization)



Weilin Cong, Morteza Ramezani, Mehrdad Mahdavi. On Provable Benefits of Depth in Training Graph Convolutional Networks. (*NeurIPS21*)

Optimization, **Generalization**, Privacy, Model design

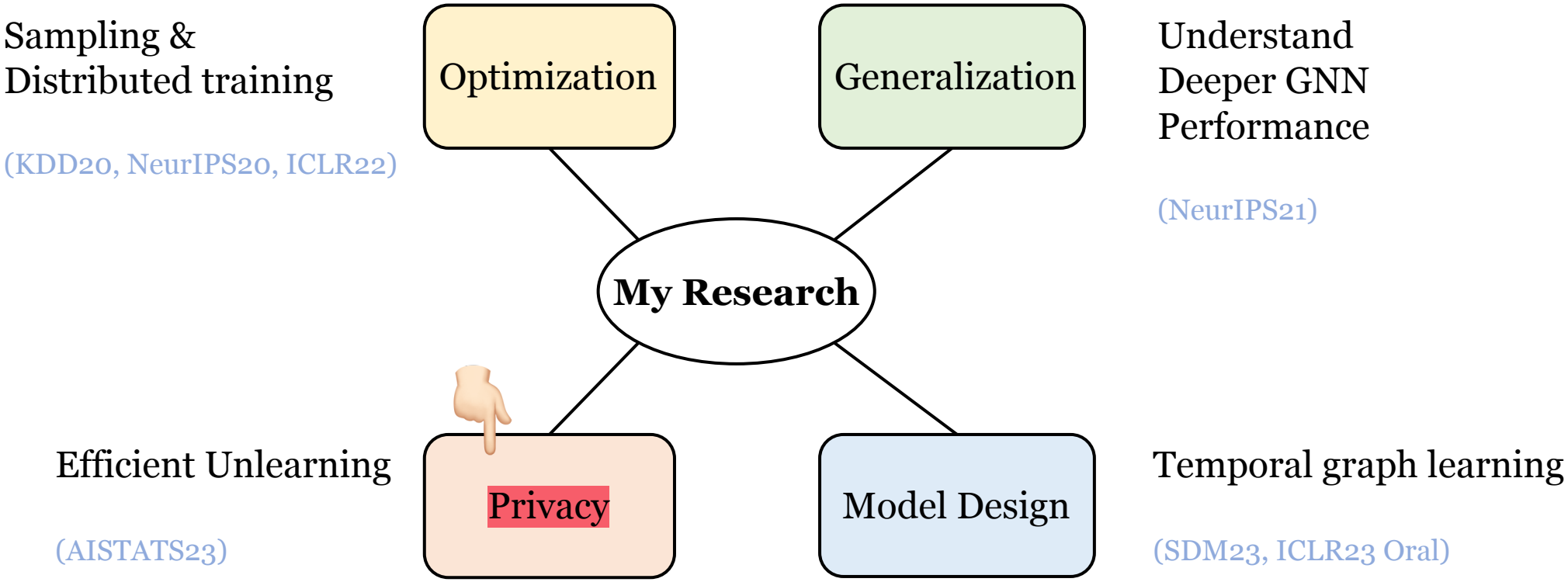
- Review theoretical analysis on over-smoothing papers, we mathematically show that over-smoothing is mainly an artifact of theoretical analysis and the assumptions made in analysis that never hold in practice;
- The "assumptions" such as:
  - GNN is linear with single weight matrix but many graph convolutions
  - (singular value of weight parameters) × (singular value of graph Laplacian) < 1
    < 2 → matrix concentration 0.99 ≈ 1 → real-world graphs are sparse
- We provide different view by analysis the impact of GNN structure on the generalization. We use uniform stability for theoretical analysis: APPNP ≤ GCNII ≤ GCN ≤ ResGCN



Weilin Cong, Morteza Ramezani, Mehrdad Mahdavi. On Provable Benefits of Depth in Training Graph Convolutional Networks. (*NeurIPS21*)

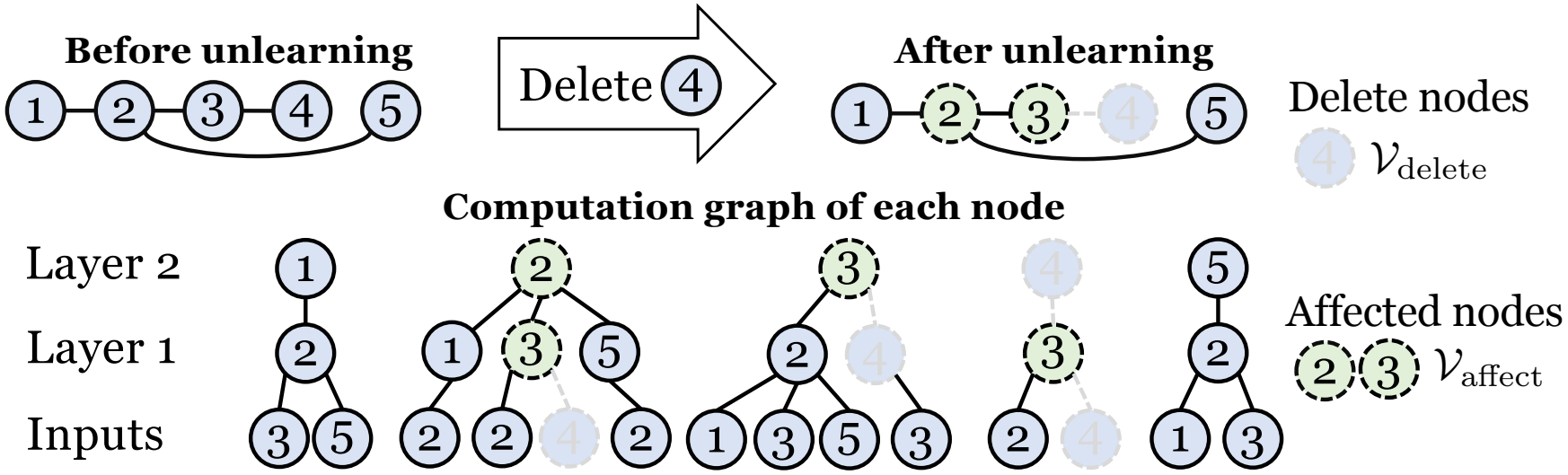
Overview on research

- During my PhD study, my research focuses on fundamental machine learning problems on graph structured data



Optimization, Generalization, **Privacy**, Model design

- Graph representation unlearning (i.e., selected forgetting) is challenging due to node dependency.

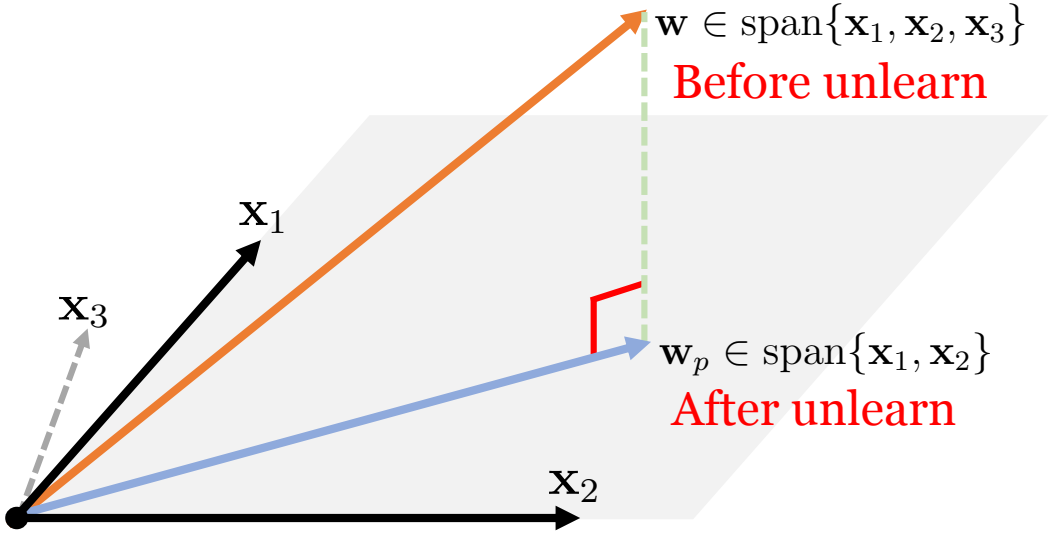


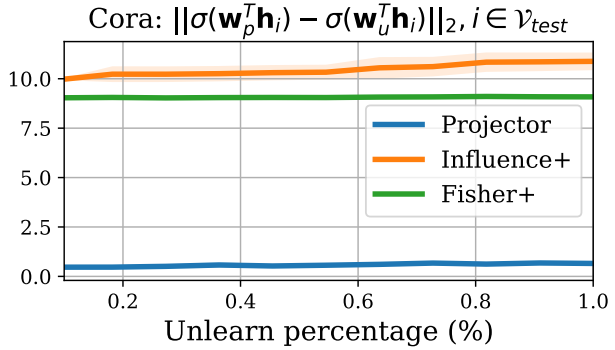
- Existing unlearning methods are designed for setting where loss function can be decomposed over individual training samples.

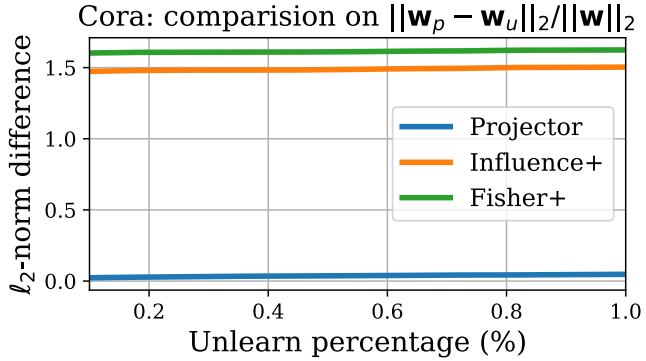
Weilin Cong, Mehrdad Mahdavi. Efficiently Forgetting What You Have Learned in Graph Representation Learning via Projection. (*AISTATS23*)

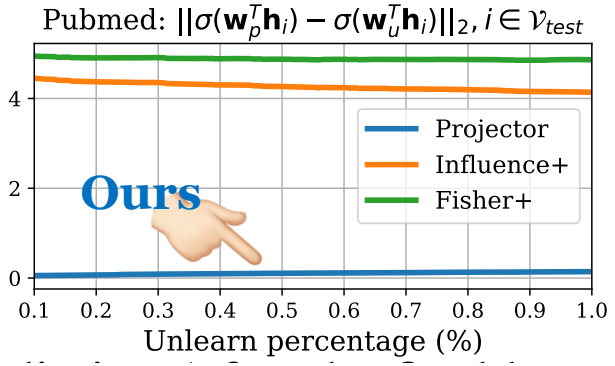
Optimization, Generalization, **Privacy**, Model design

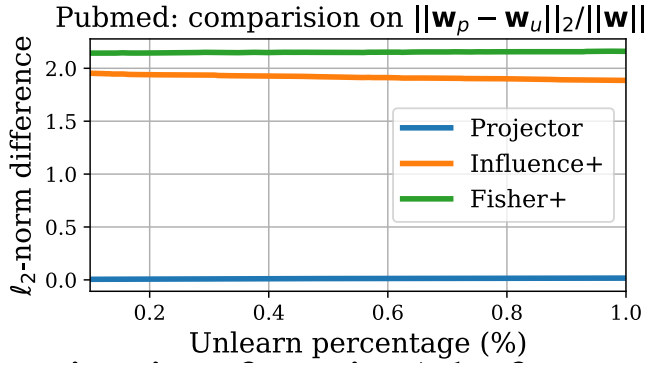
- (Algorithm) We propose to unlearn by projecting the weight parameters of the pre-trained model onto a subspace that is irrelevant to features of the nodes to be forgotten.
- (Theory) We theoretically upper bound the distance between the unlearned weight parameters to the weight parameters obtained by re-training on the new dataset without the deleted nodes.







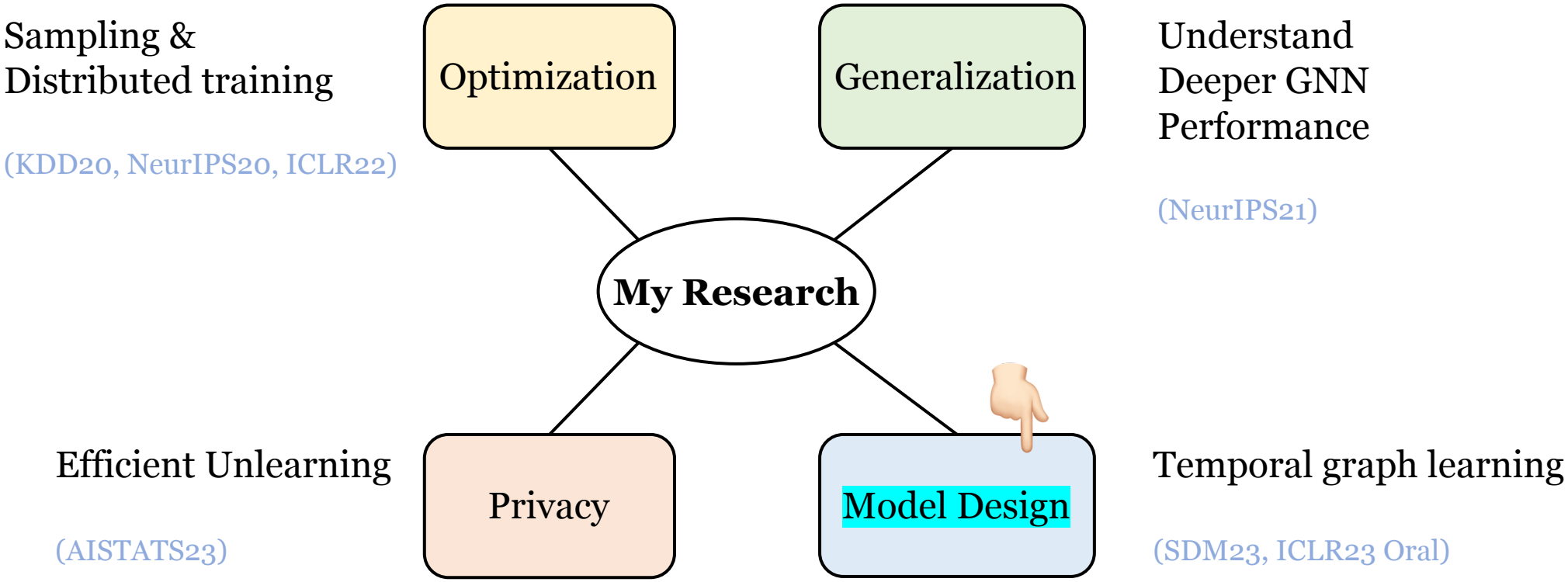




Weilin Cong, Mehrdad Mahdavi. Efficiently Forgetting What You Have Learned in Graph Representation Learning via Projection. (*AISTATS23*)

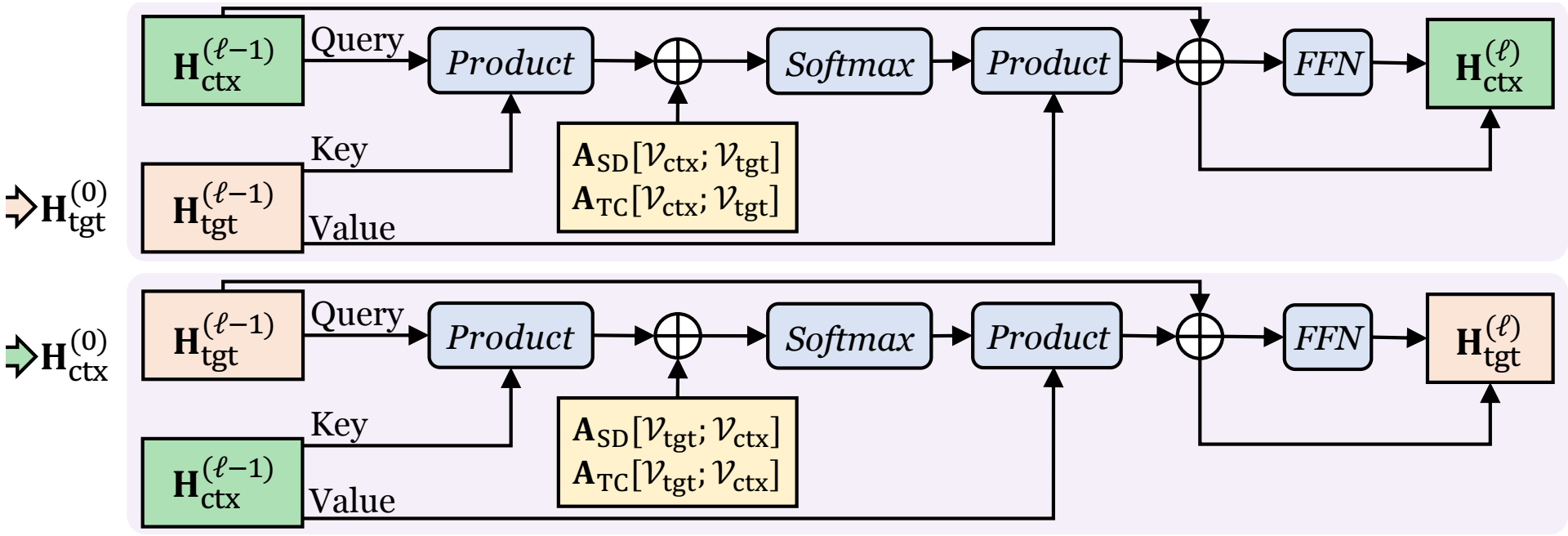
Overview on research

- During my PhD study, my research focuses on fundamental machine learning problems on graph structured data



Optimization, Generalization, Privacy, **Model design**

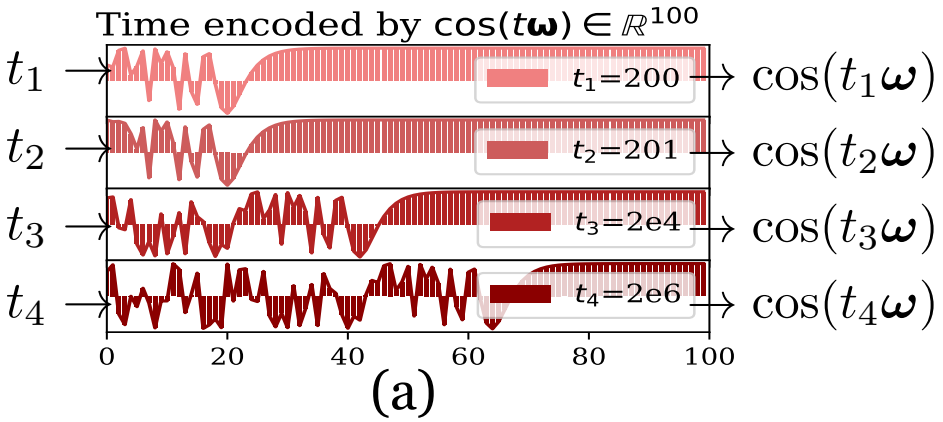
- Use Transformer for temporal graph learning.
- (Algorithm) We propose a scalable Transformer-like temporal graph learning method
- (Theory) To improve the generalization ability, we introduce self-supervised pre-training task and show that jointly optimizing them results in a smaller Bayesian error via an information theoretic analysis



Weilin Cong, Yanhong Wu, Yuandong Tian, Mengting Gu, Yinglong Xia, Jason Chen, Mehrdad Mahdavi
DyFormer : A Scalable Dynamic Graph Transformer with Provable Benefits on Generalization Ability. (*SDM23*)

Optimization, Generalization, Privacy, **Model design**

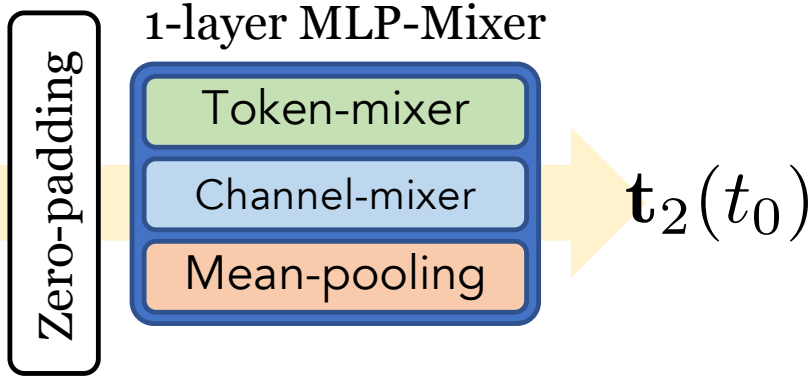
- Design neural architecture for temporal graph learning (TGL).
- (Experiments) RNN and self-attention mechanism (SAM) are the de facto standard for TGL. Although both RNN and SAM could lead to good performance, in practice neither of them is always necessary.
- (Algorithm) We propose a conceptually and technically simple architecture, which attains an outstanding performance with faster convergence and better generalization ability.



(a)

$$\begin{pmatrix} [\cos((t_0 - t_1) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{1,2}^{\text{link}}(t_1)] \\ [\cos((t_0 - t_3) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{2,4}^{\text{link}}(t_3)] \\ [\cos((t_0 - t_4) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{2,4}^{\text{link}}(t_4)] \\ [\cos((t_0 - t_5) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{1,2}^{\text{link}}(t_5)] \end{pmatrix}$$

(b)



Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, Mehrdad Mahdavi
Do We Really Need Complicated Model Architectures For Temporal Networks? (*ICLR23 Oral*)

Summarization on research visions

- What makes my research different from others?
- (1) Fundamental problems > Specific applications
- (2) Theoretical analysis & Empirical evaluation & Model design
- (3) Rethink the apparent consensus > Following the apparent consensus:
  - (Generalization) “Over-smoothing” not necessarily happen in practice, performance degradation issue is due to generalization
  - (Privacy) Differential privacy (DP)-based unlearning might fail and results in poor performance, we propose a theory guided Projection-based unlearning method
  - (Model design) Intuitively, RNN and self-attention is suitable for temporal graph. However, we found that RNN and self-attention free method could achieve better performance.

DO WE REALLY NEED COMPLICATED MODEL ARCHITECTURES FOR TEMPORAL NETWORKS?

Weilin Cong
Penn State
weilin@psu.edu

Si Zhang
Meta
sizhang@meta.com

Jian Kang
University of Illinois at Urbana-Champaign
jiank2@illinois.edu

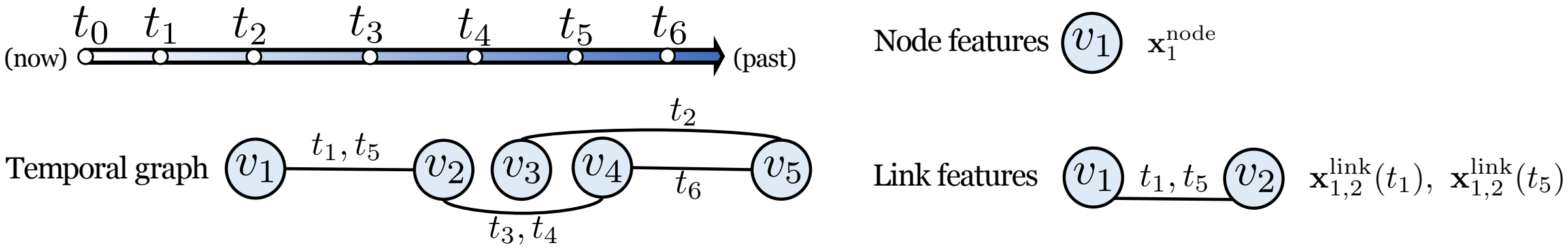
Baichuan Yuan & Hao Wu & Xin Zhou
Meta
{bcyuan,haowu1,markzhou}@meta.com

Hanghang Tong
University of Illinois at Urbana-Champaign
htong@illinois.edu

Mehrdad Mahdavi
Penn State
mzm616@psu.edu

Background

- Temporal network structure:



- Goal: prediction if v_i, v_j interact at t_0 based on all the available temporal graph information during $\{t_1, \ldots, t_6\}$
- Application: recommender system, traffic prediction

Motivation

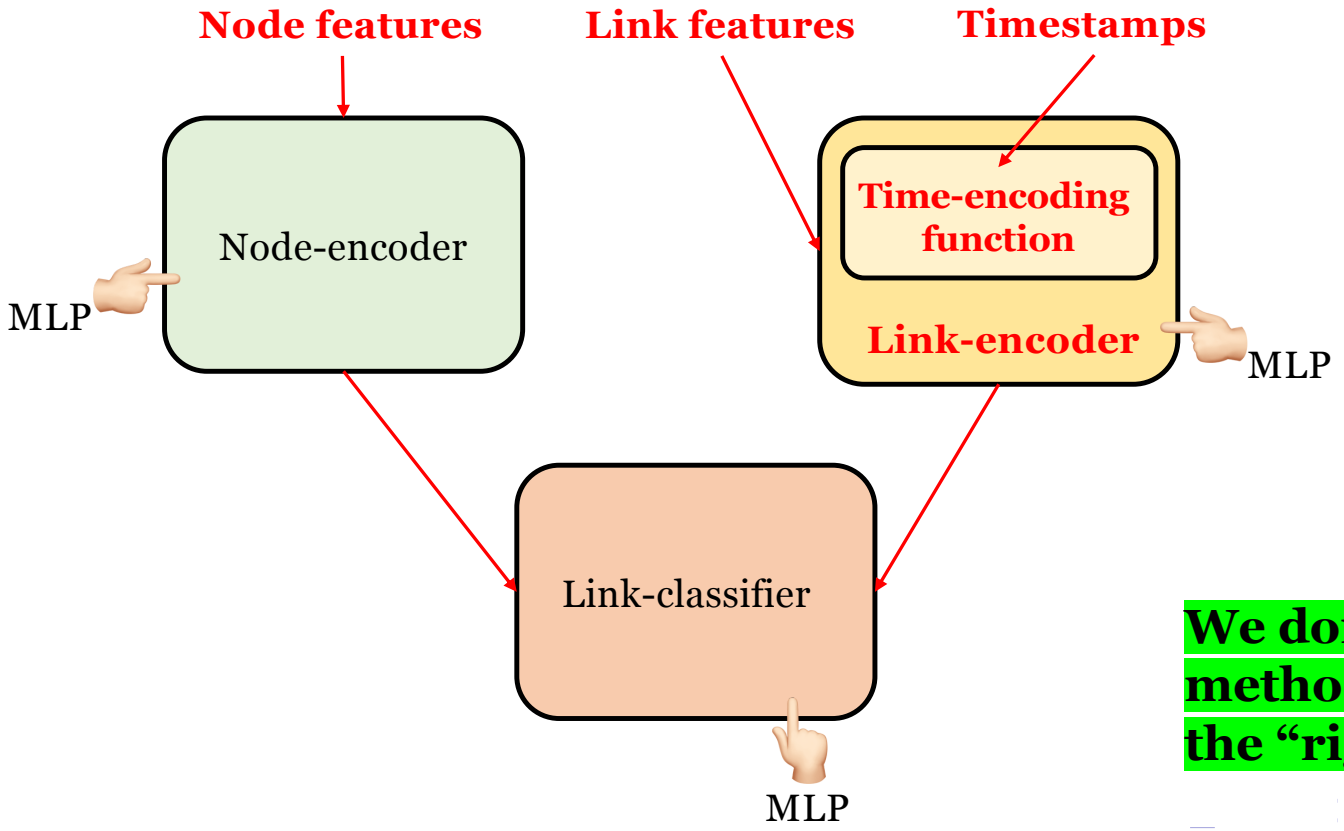
- **Existing works**: RNN and self-attention mechanism (SAM) are the de facto standard for temporal graph learning, e.g.,
  - JODIE: Inputs → RNN + Memory blocks
  - TGAT: Inputs → Self-attention mechanism
  - TGN: Inputs → RNN + Memory blocks → Self-attention mechanism

- Indeed, such architecture design matches our intuition

- It has the following drawbacks:
  - These methods are conceptually and technically complicated with advanced model architecture, which is hard to implement
  - Hard to understand which parts of the model truly contribute to its success, and whether these components are indispensable

Rethinking …

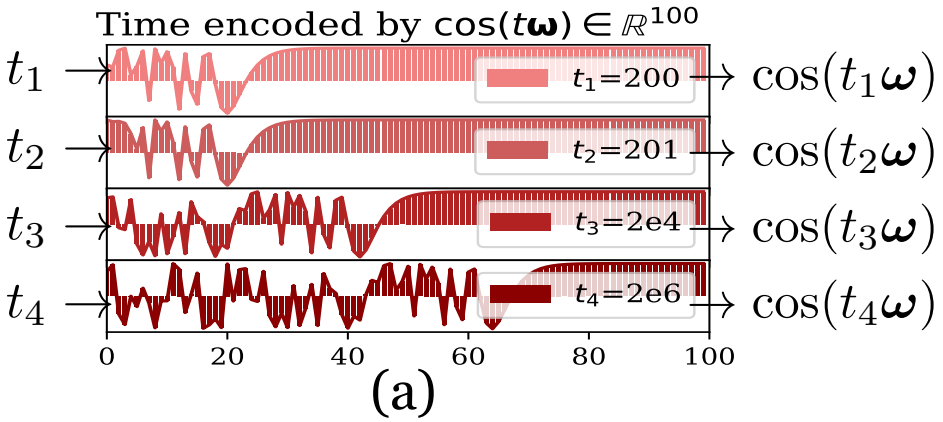
- **Q: Are RNN and self-attention indispensable for TGL?**
- A: Not really …
  - We propose **GraphMixer** that based entirely on the MLPs and neighbor mean-pooling;
  - **GraphMixer** achieves SOTA performance with even smaller computation cost and number of parameters



We don't need complicated method if we could select the "right" input data

Link-encoder, Node-encoder, Link-classifier

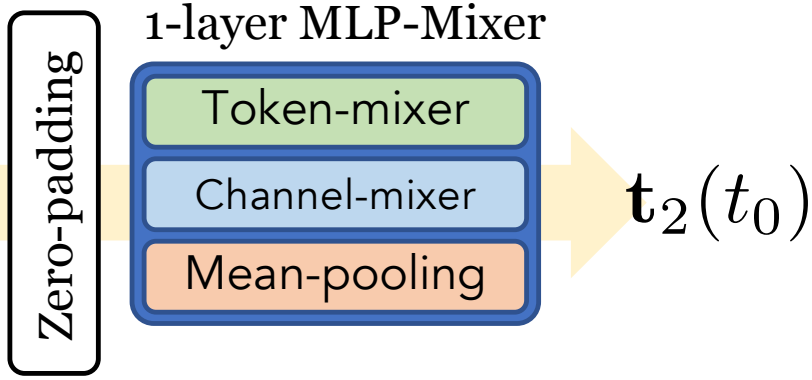
- Design to summarize the temporal link information (link timestamps and link features) with each node sorted by timestamps.
- To distinguish different timestamps, we introduce our time-encoding function $\cos(t\boldsymbol{\omega})$ to encode each timestamps into d-dimensional vector, where $\boldsymbol{\omega} = \{\alpha^{-(i-1)/\beta}\}_{i=1}^{d}$ is fixed not trainable



(a)

$$\begin{pmatrix} [\cos((t_0 - t_1) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{1,2}^{\text{link}}(t_1)] \\ [\cos((t_0 - t_3) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{2,4}^{\text{link}}(t_3)] \\ [\cos((t_0 - t_4) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{2,4}^{\text{link}}(t_4)] \\ [\cos((t_0 - t_5) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{1,2}^{\text{link}}(t_5)] \end{pmatrix}$$

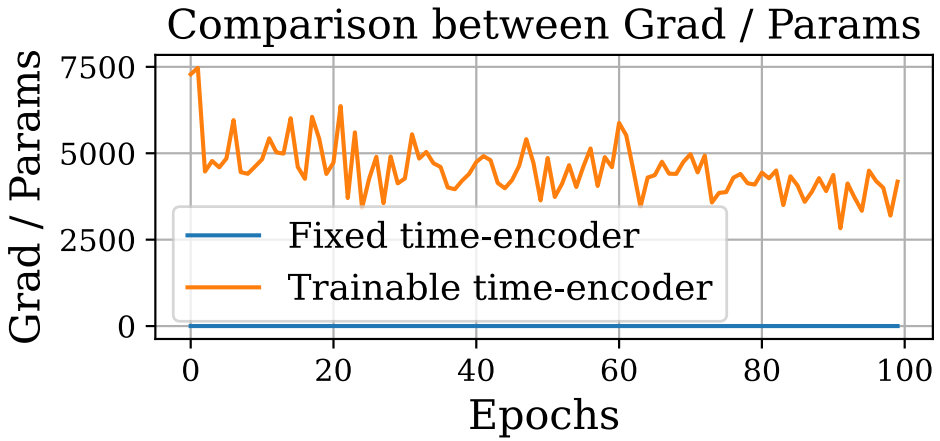
(b)

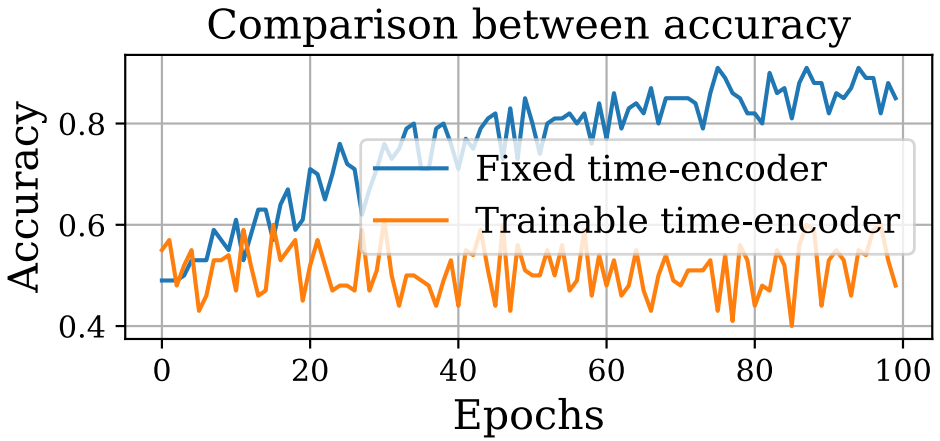


- Our time-encoding function enjoys two properties:
  - Similar timestamps have similar time-encodings (e.g., the plot of t_1, t_2)
  - The larger the timestamps, the later the values in time-encodings convergence to +1. (e.g., the plot of t_1, t_3 and t_1, t_4)

Quick experiment on time-encoding function

- Existing works leverage a trainable time-encoding function $z(t) = \cos(tw + b)$ to represent timestamps.
- However, trainable time-encoding function could cause instability during training because its gradient $\frac{\partial \cos(tw+b)}{\partial w} = t \times \sin(tw + b)$ scales proportional to the timestamps

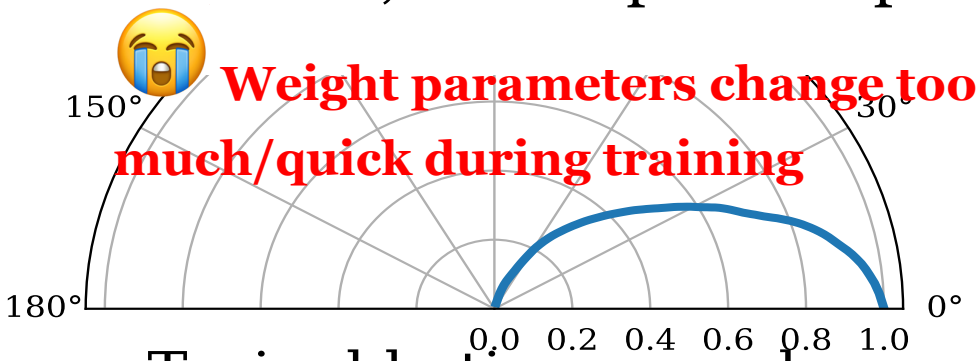




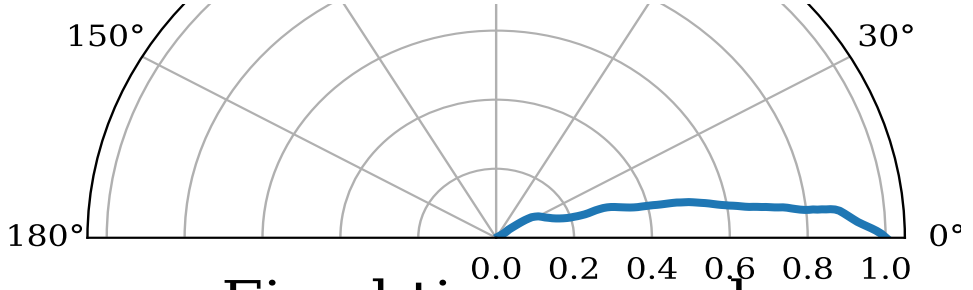
- Setup: given any $t_1, t_2 \in [0, 10^6]$, our goal is to classify if $t_1 > t_2$ by learning a linear classifier on $[z(t_1) \parallel z(t_2)]$
- Results: we give the gradient norm (left figure) and accuracy (right figure) at each iteration. We can observe that trainable time-encoding function suffer from training instability issue.

Quick experiment on time-encoding function

- Meanwhile, we compare the parameter trajectories of the two models:



Trainable time-encoder



Fixed time-encoder

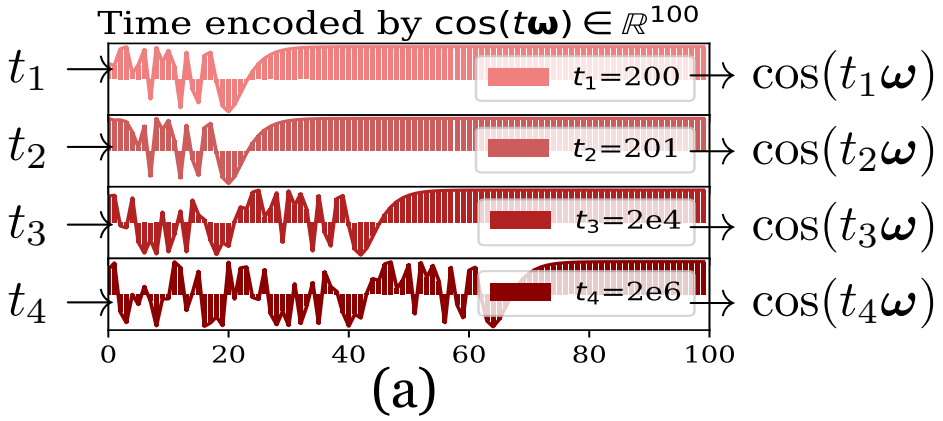
- We observe that the change of parameters on the trainable time-encoding function is drastically larger than our fixed version
- A huge change in weight parameters could deteriorate the model performance

Table: Comparison on average precision score with fixed/trainable time encoding function. "Trainable" → "Fixed".

	Reddit	Wiki	MOOC	LastFM	GDELT-ne	GDELT-e
JODIE	99.30 → **99.76**	98.81 → **99.00**	99.16 → **99.17**	67.51 → **79.89**	97.13 → **98.23**	96.96 → **96.96**
TGAT	98.66 → **99.48**	96.71 → **98.55**	98.43 → **99.33**	54.77 → **76.26**	84.30 → **92.31**	**96.96** → 96.28
TGN	99.80 → **99.83**	**99.55** → 99.54	99.62 → **99.62**	82.23 → **87.58**	98.15 → **98.25**	96.04 → **97.34**

Link-encoder, Node-encoder, Link-classifier

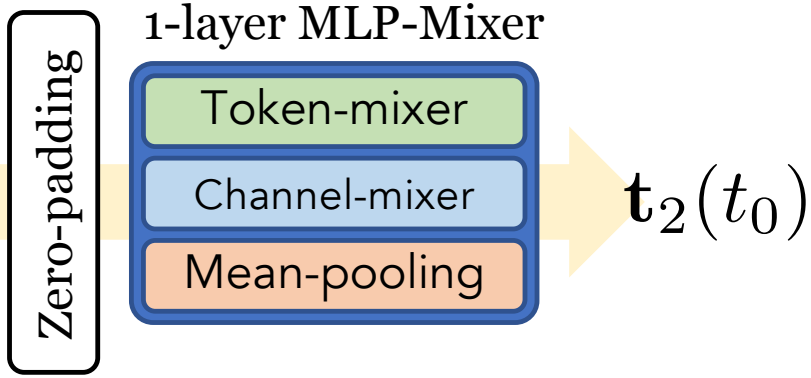
- To summarize the temporal link information, we use 1-layer MLP



(a)

$$\begin{pmatrix} [\cos((t_0 - t_1) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{1,2}^{\text{link}}(t_1)] \\ [\cos((t_0 - t_3) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{2,4}^{\text{link}}(t_3)] \\ [\cos((t_0 - t_4) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{2,4}^{\text{link}}(t_4)] \\ [\cos((t_0 - t_5) \times \boldsymbol{\omega}) \,||\, \mathbf{x}_{1,2}^{\text{link}}(t_5)] \end{pmatrix}$$

(b)



- To summarize the temporal link information of a node,
  - Firstly, we encode timestamps by our time-encoding function, then concatenate it with its corresponding link features
  - Then, we stack all the outputs into a big matrix and zero-pad to the fixed length K
  - Finally, we use a 1-layer MLP-mixer with mean-pooling to compress it into a single vector

Quick experiments on link-encoder

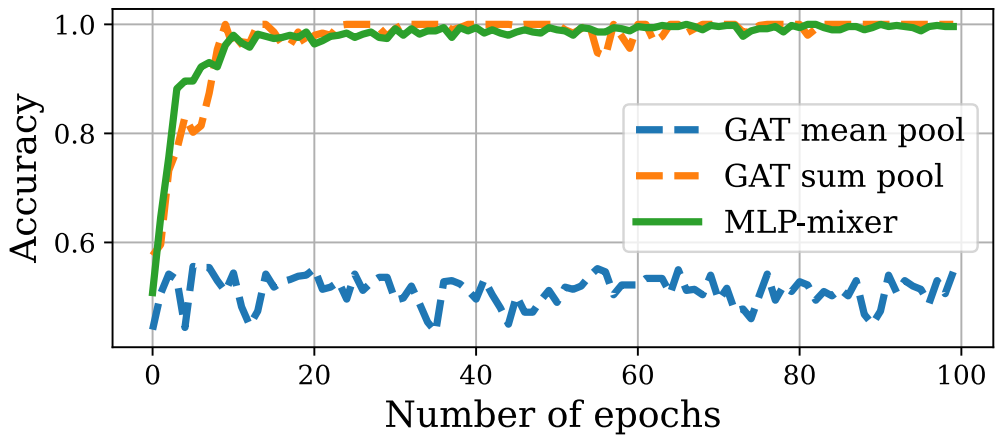
- Q2: Can we replace the MLP-mixer in link-encoder with self-attention?
- We test by replacing the MLP-mixer in link-encoder with
  - Full self-attention / 1-hop self-attention
  - Sum pooling / mean-pooling

Link-info encoder with		Reddit	Wiki	MOOC	LastFM	GDELT-ne
(Default) MLP-mixer	+ Zero-padding	**99.93**	**99.85**	**99.91**	**96.31**	**98.39**
Full self-attention	+ Sum pooling	99.81	98.19	99.55	93.97	98.28
	+ Mean pooling	99.00	98.05	99.31	89.15	97.13
1-hop self-attention	+ Sum pooling	99.81	98.01	99.30	93.69	98.16
	+ Mean pooling	98.94	97.29	98.96	72.32	97.09

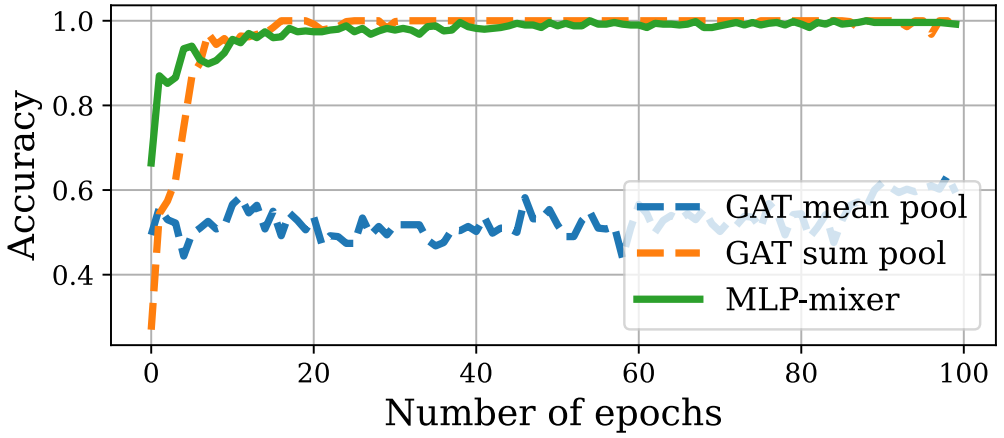
- Performance drop when using self-attention:
  - The best performance is achieved when using MLP-mixer with zero-padding;
  - The model performance drop slightly when using self-attention with sum-pooling (2^{nd} and 4^{th} row)
  - The performance drop significantly when using self-attention with mean pooling (3^{rd} and 5^{th} row)

Quick experiments on link-encoder

- Q3: Self-attention with mean-pooling has a weaker model performance?
- It cannot distinguish "temporal sequence with identical link timestamps and link features". For example, it cannot distinguish $[a_1, a_1]$ and $[a_1]$;
- It cannot explicitly capture "the length of temporal sequence". For example, it cannot distinguish if $[a_1, a_2]$ is longer than $[a_3]$;



(a) e.g., classify if $[a_1, a_1] = [a_1]$

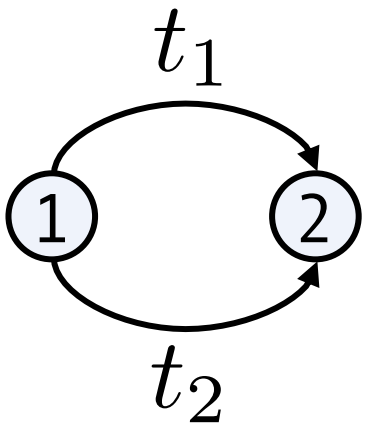


(b) e.g., classify if $\text{size}[a_1, a_2] > \text{size}[a_3]$

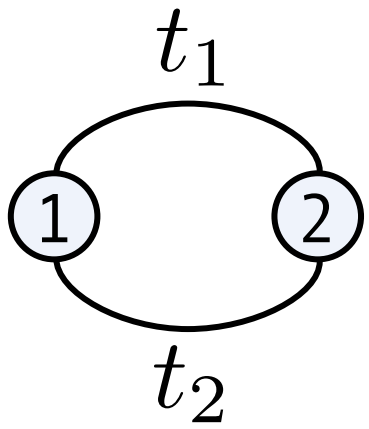
These properties are very important to understand how frequent a node interacts with other nodes → related to the input data.

Different inputs between ours and others

- Temporal graph as undirected vs directed graph
  - Most of the existing works consider temporal graphs as directed graphs with information only flows from the source nodes to the destination nodes
  - However, we consider the temporal graph as undirected graph
  - By doing so, if two nodes are frequently connected in the last few timestamps, the "most recent 1-hop neighbors" sampled for the two nodes on the "undirected" temporal graph would be similar



$\mathcal{N}(v_1) = \{v_2(t_1), v_2(t_2)\}$
$\mathcal{N}(v_2) = \{\}$



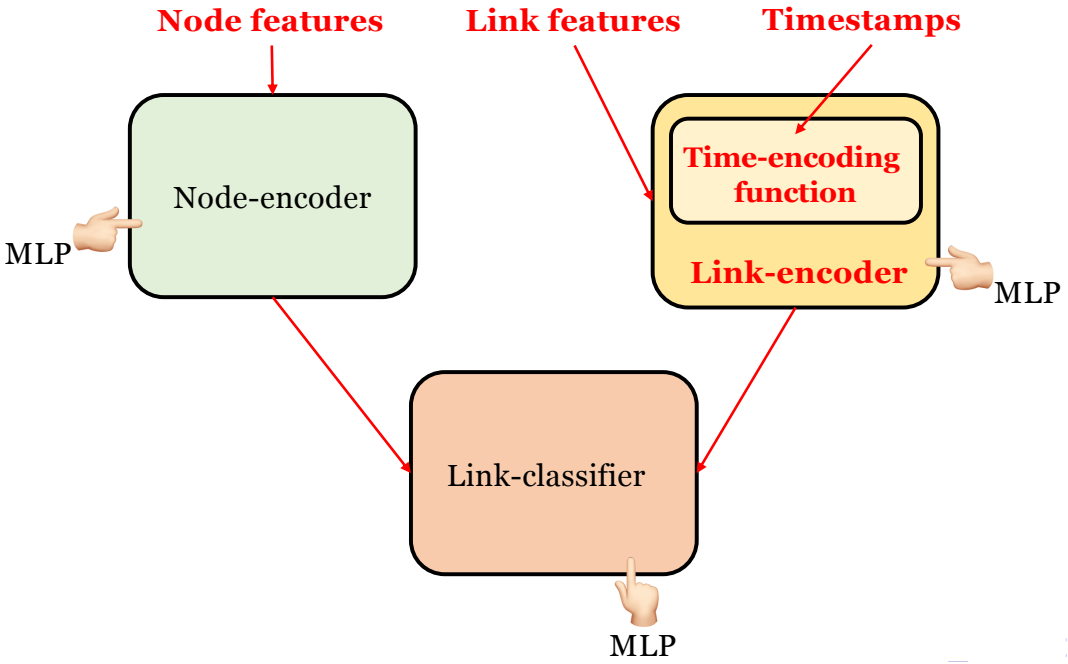
$\mathcal{N}(v_1) = \{v_2(t_1), v_2(t_2)\}$
$\mathcal{N}(v_2) = \{v_1(t_1), v_1(t_2)\}$

- Intuitively, if two nodes are frequently connected in the last few timestamps, they are also likely to be connected in the recent future

Link-encoder, Node-encoder, Link-classifier

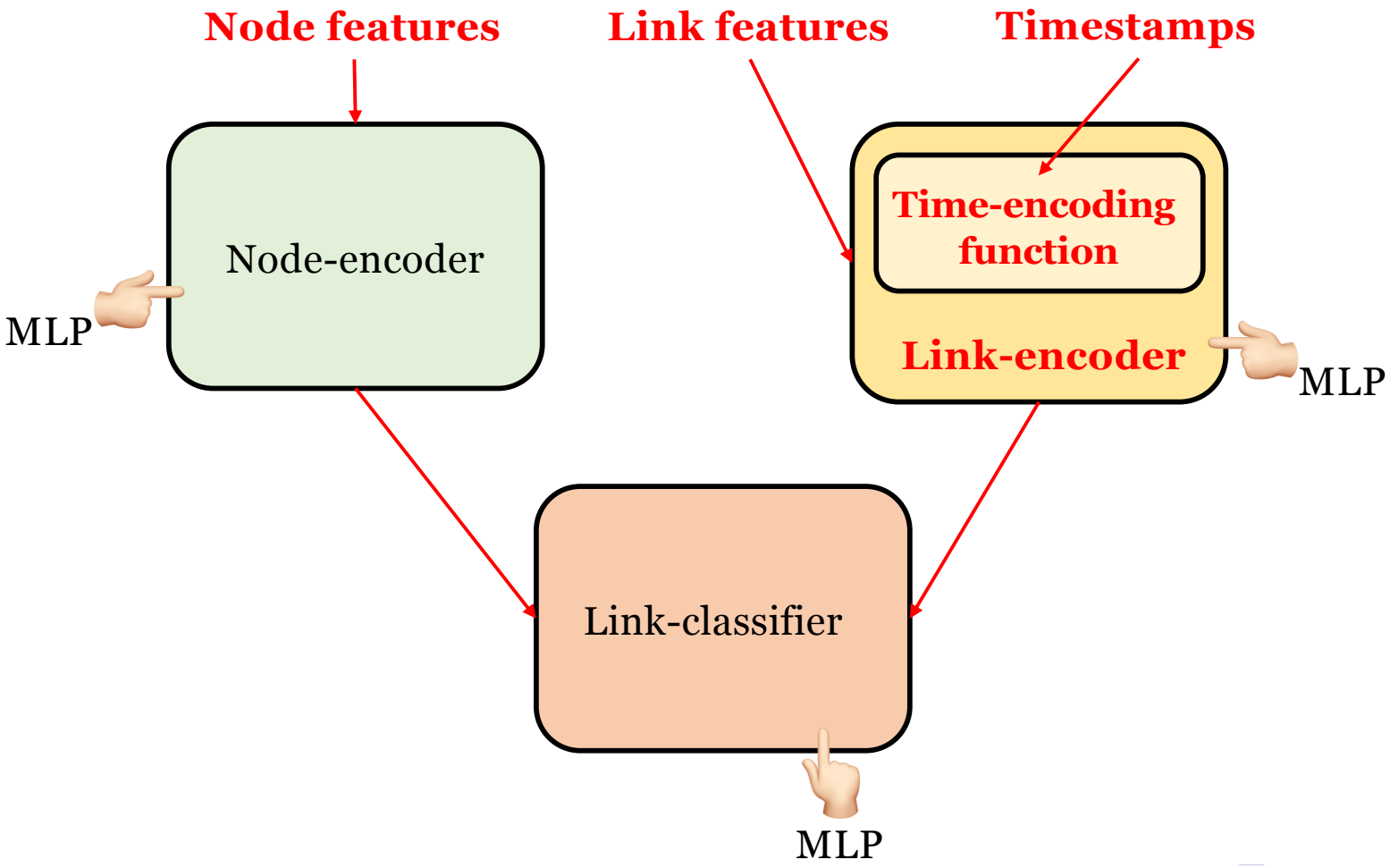
- Node encoder is designed to capture the node identity and node feature information via neighbor mean-pooling (i.e., 1-hop SIGN features)
- $\mathcal{N}(v_i; t, t_0)$ as 1-hop neighbor of node v_i with link timestamps from t to t_0
- Then, the node information features are computed based on the 1-hop neighbor by

$$\boldsymbol{s}_i(t_0) = \boldsymbol{x}_i^{node} + Mean\{\boldsymbol{x}_j^{node} \mid v_j \in \mathcal{N}(v_i; t_0 - T, t_0)\}$$



Link-encoder, Node-encoder, Link-classifier

- Link-classifier is computed by applying 2-layer MLP model on the concatenated output of node-encoder and link-encoder



Experiments

- GraphMixer achieves outstanding performance
- Two variant of GraphMixer:
  - GraphMixer-L: only use link-encoder + link-classifier
  - GraphMixer-N: only use node-encoder + link-classifier

Larger average time-gaps
Larger average node-degree
Larger maximum timestamps

Table: Comparison on the average precision score for link prediction.

	Reddit L, T	Wiki L, T	MOOC T	LastFM T	GDELT L, N, T	GDELT-ne T	GDELT-e N, T
JODIE	99.30 ± 0.01	98.81 ± 0.01	99.16 ± 0.01	67.51 ± 0.87	98.27 ± 0.02	97.13 ± 0.02	96.96 ± 0.02
DySAT	98.52 ± 0.01	96.71 ± 0.02	98.82 ± 0.02	76.40 ± 0.77	98.52 ± 0.02	82.47 ± 0.13	97.25 ± 0.02
TGAT	99.66 ± 0.01	97.75 ± 0.02	98.43 ± 0.01	54.77 ± 1.01	98.25 ± 0.02	84.30 ± 0.10	96.96 ± 0.02
TGN	99.80 ± 0.01	99.55 ± 0.01	99.62 ± 0.01	82.23 ± 0.50	98.15 ± 0.02	97.13 ± 0.02	96.04 ± 0.02
CAWs-mean	98.43 ± 0.02	97.72 ± 0.03	62.99 ± 0.87	76.35 ± 0.08	95.11 ± 0.12	69.20 ± 0.10	91.72 ± 0.19
CAWs-attn	98.51 ± 0.02	97.95 ± 0.03	63.07 ± 0.82	76.31 ± 0.10	95.06 ± 0.11	69.54 ± 0.19	91.54 ± 0.22
TGSRec	95.21 ± 0.08	91.64 ± 0.12	83.62 ± 0.34	76.91 ± 0.87	97.03 ± 0.61	97.03 ± 0.61	97.03 ± 0.61
APAN	99.24 ± 0.02	98.14 ± 0.01	98.70 ± 0.98	69.39 ± 0.81	95.96 ± 0.10	97.38 ± 0.23	96.77 ± 0.18
GraphMixer-L	99.84 ± 0.01	99.70 ± 0.01	99.81 ± 0.01	95.50 ± 0.03	98.99 ± 0.02	96.14 ± 0.02	98.99 ± 0.02
GraphMixer-N	99.24 ± 0.01$^\natural$	90.33 ± 0.01$^\natural$	97.35 ± 0.02$^\natural$	63.80 ± 0.03$^\natural$	94.44 ± 0.02	96.00 ± 0.02$^\natural$	98.81 ± 0.02$^\natural$
GraphMixer	99.93 ± 0.01$^\natural$	99.85 ± 0.01$^\natural$	99.91 ± 0.01$^\natural$	96.31 ± 0.02$^\natural$	98.89 ± 0.02	98.39 ± 0.02$^\natural$	98.22 ± 0.02$^\natural$

Experiments

- GraphMixer enjoys better convergence and generalization ability

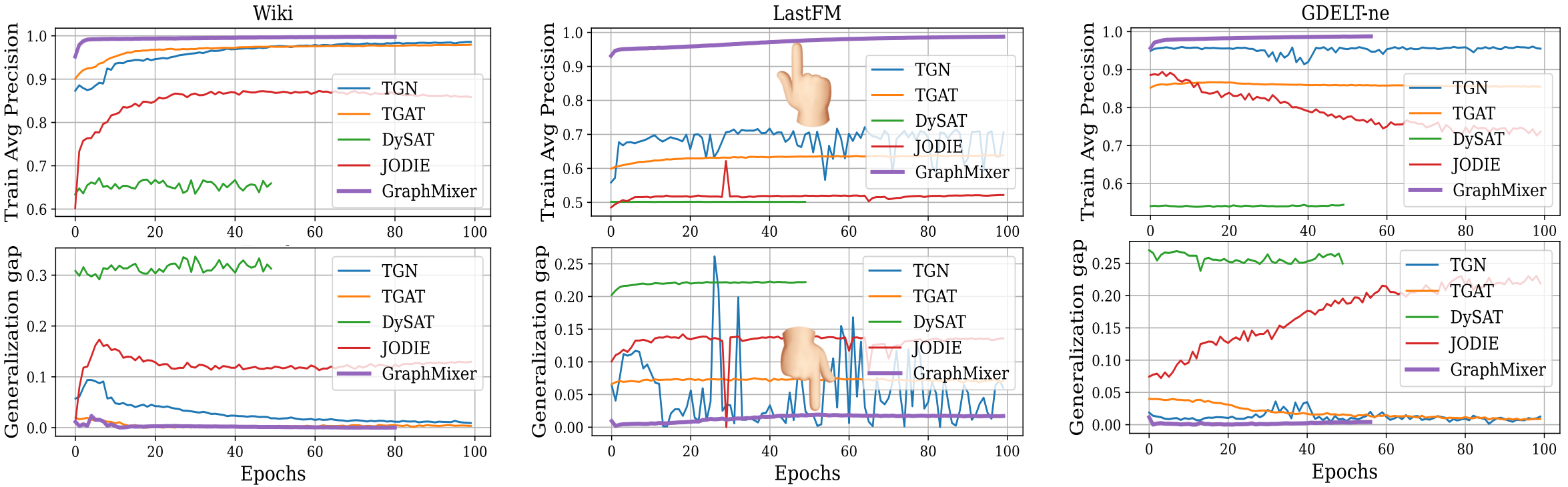
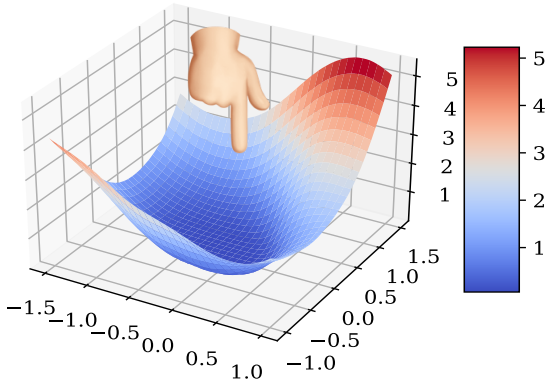


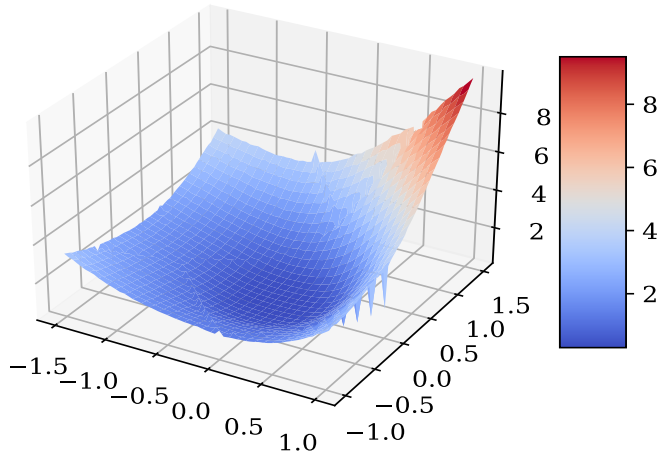
Figure: Comparison on the training set average precision and generalization gap for the first 100 training epochs.

Experiments

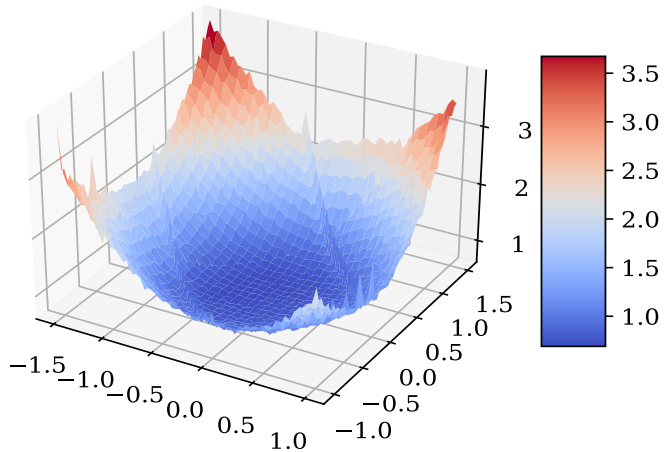
- Optimization landscape



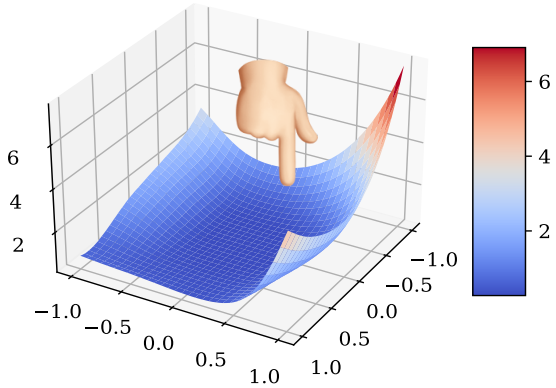
(a) GraphMixer on Wiki



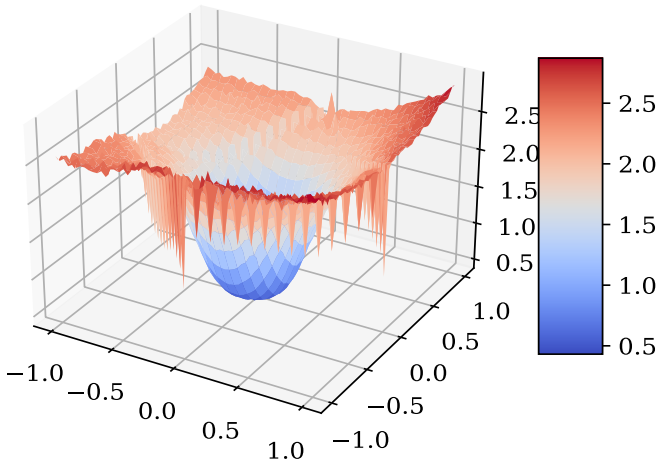
(b) TGAT on Wiki



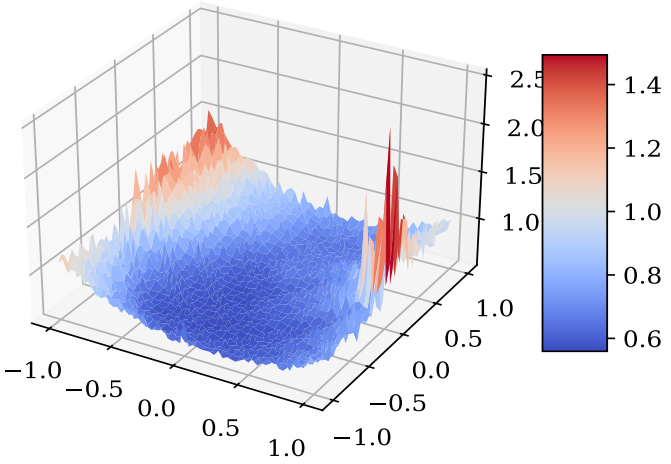
(c) TGN on Wiki



(d) GraphMixer on GDELT-e



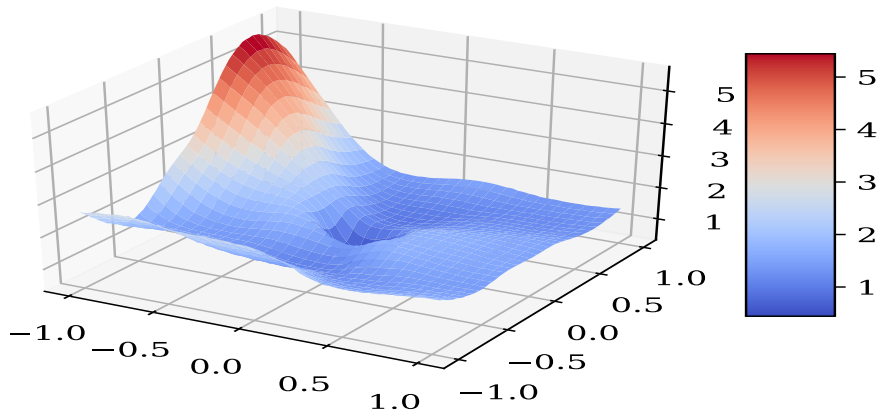
(e) TGAT on GDELT-e



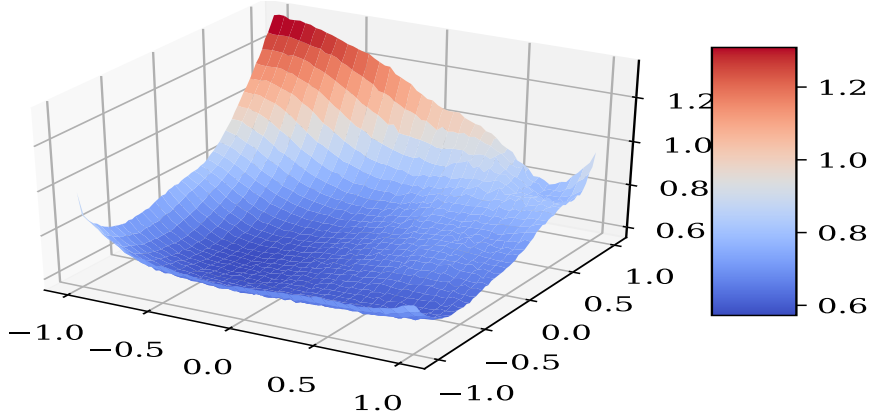
(f) TGN on GDELT-e

Experiments

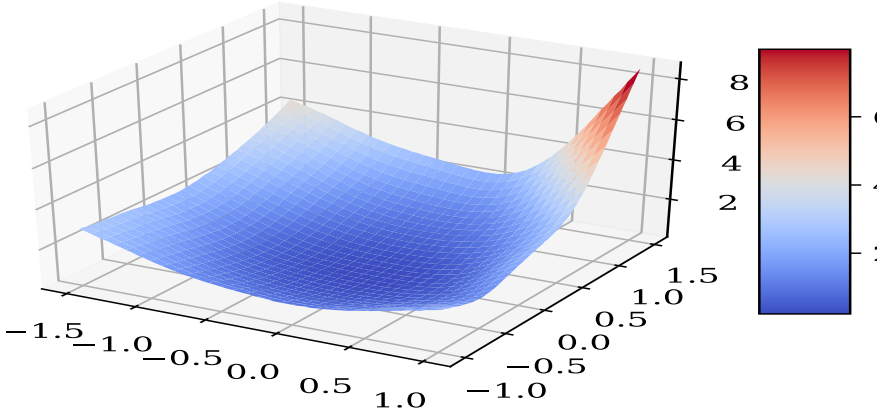
- Optimization landscape: existing methods + fixed time-encoding function



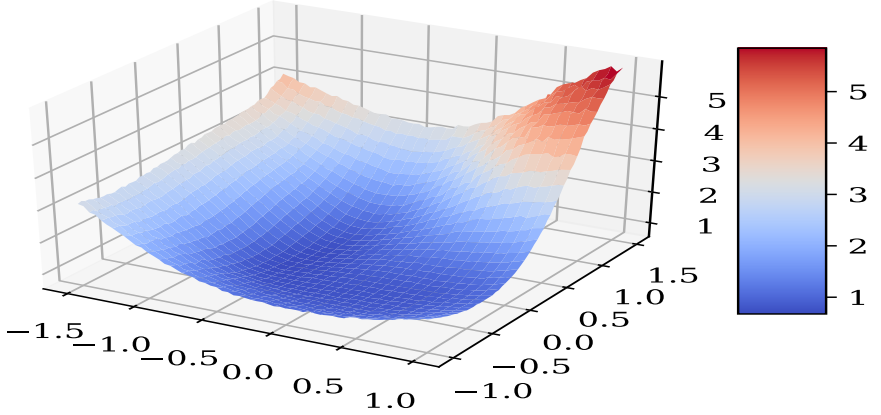
(a) TGAT on GDELT-e



(b) TGN on GDELT-e



(c) TGAT on Wiki

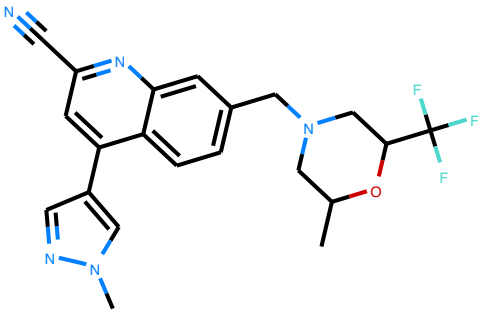


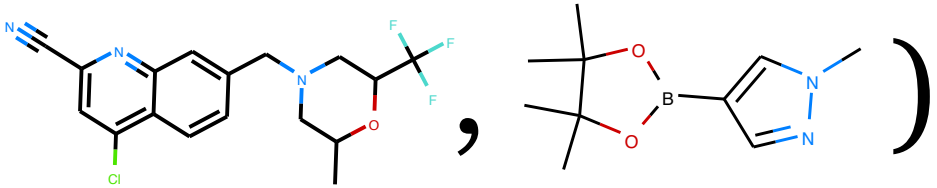
(d) TGN on Wiki

Future directions

- Understand why simple temporal graph learning method work from using generalization theory? (In progress, under-review) 🤞🏻

- Graph-level prediction tasks
  - molecule property prediction, molecule interaction prediction

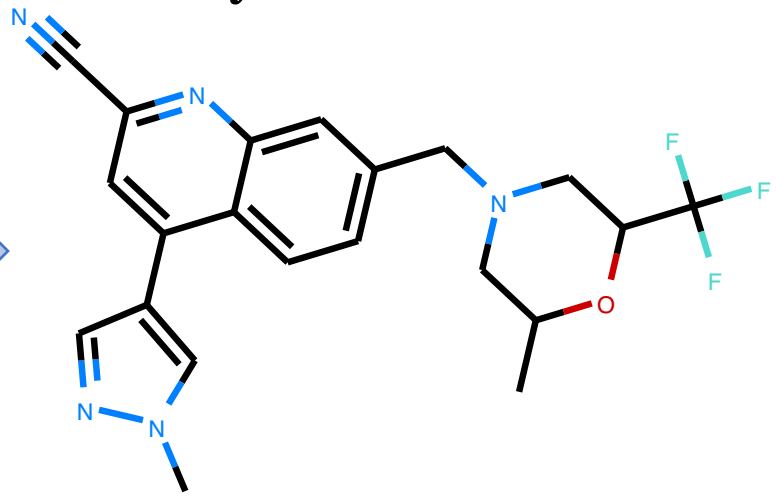
GNN( **)** → Properties?

GNN( **)** → Interact?

Future directions:

- Graph-level generation tasks
  - Application: Molecule design and discovery

Properties → **GNN** →



Q&A time

What I have presented today:

- PhD Research
  - Optimization: Sampling + Distributed learning
  - Generalization: Over-smoothing related
  - Privacy: Unlearning / Selective forgetting
  - Model design: Temporal graph learning

- Temporal graph learning
  - Link-encoder
  - Node-encoder
  - Link-classifier