Introduction to Graph Neural Networks

Yogesh Kulkarni

Principal Architect, CTO Office, Icertis, Pune, India

9 to 10 am IST, ??, 2021

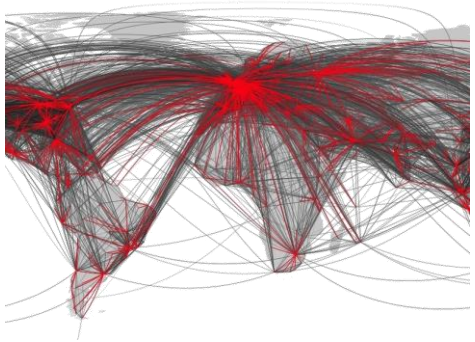
TensorFlow

User Group Pune

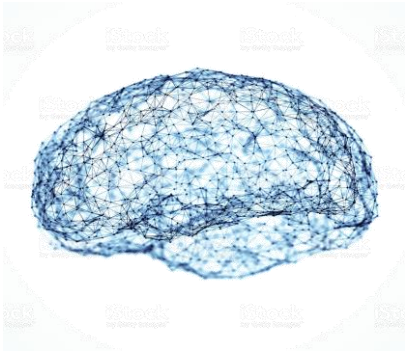
https://www.meetup.com/Tensorflow-User-Group-Pune

Data as Graphs - Explicit

Social Graphs

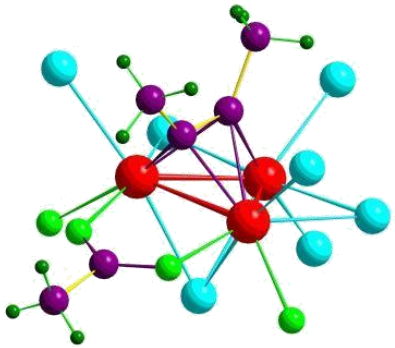


Transportation Graphs

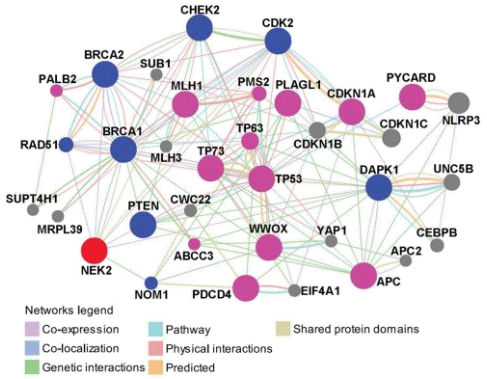


Brain Graphs

Web Graphs

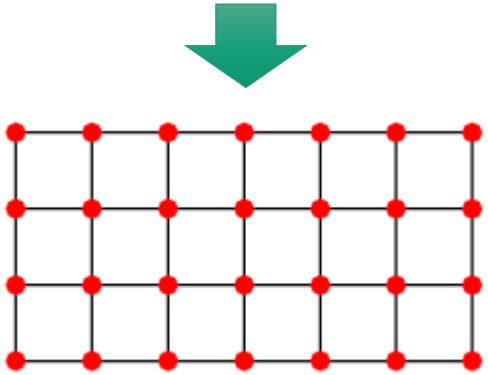


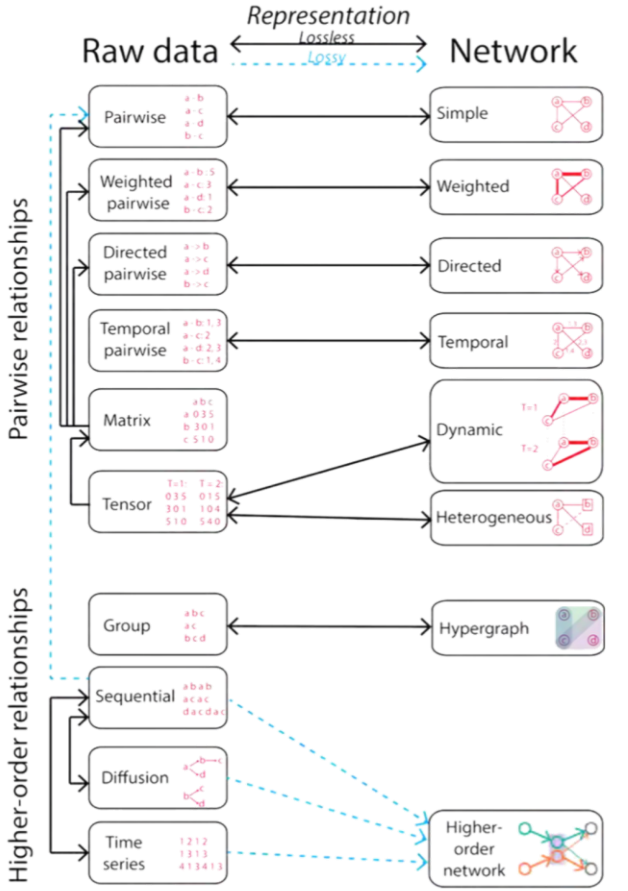
Molecular Graphs



Gene Graphs

Data as Graphs - Implicit

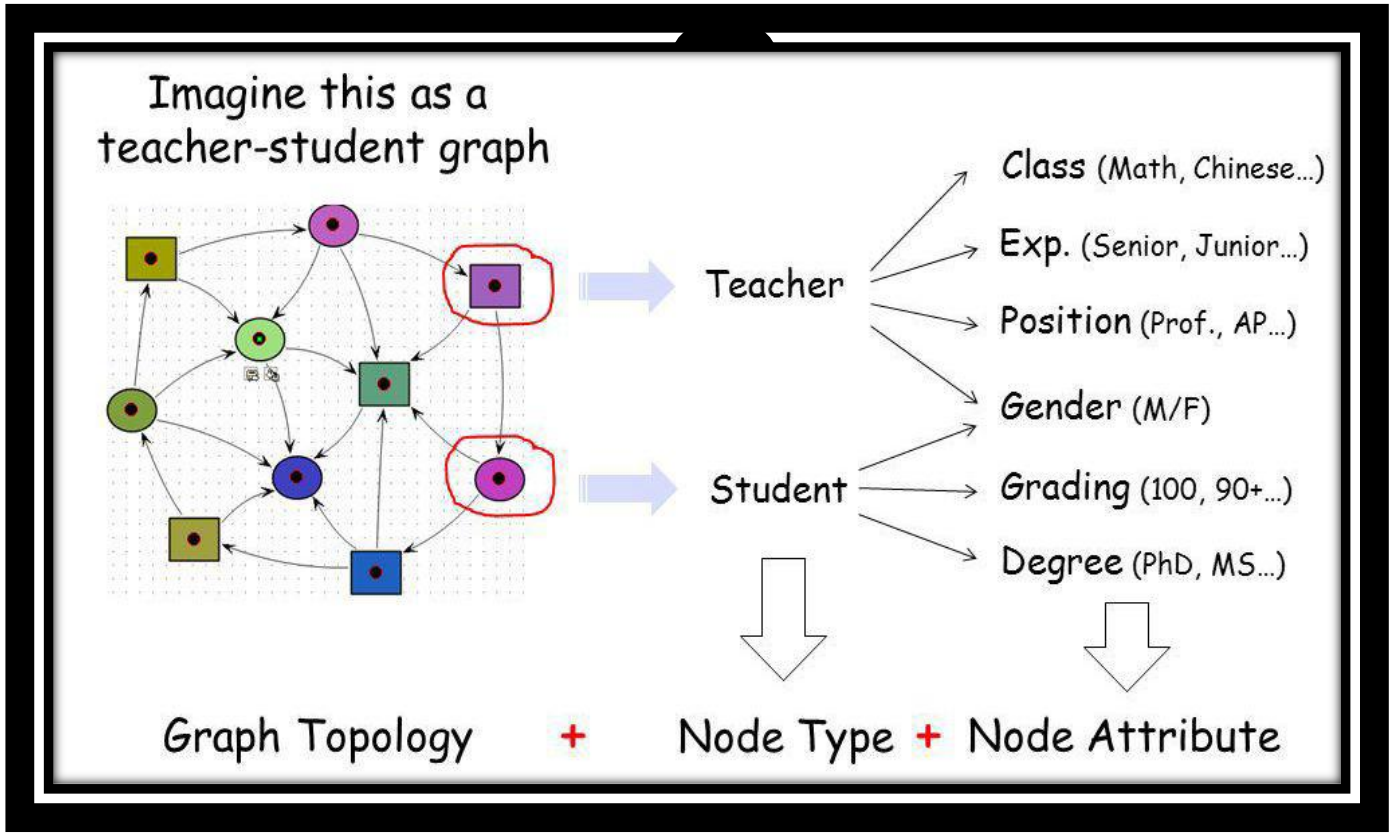




Jian Xu. Representing Big Data as Networks. PhD Dissertation, University of Notre Dame

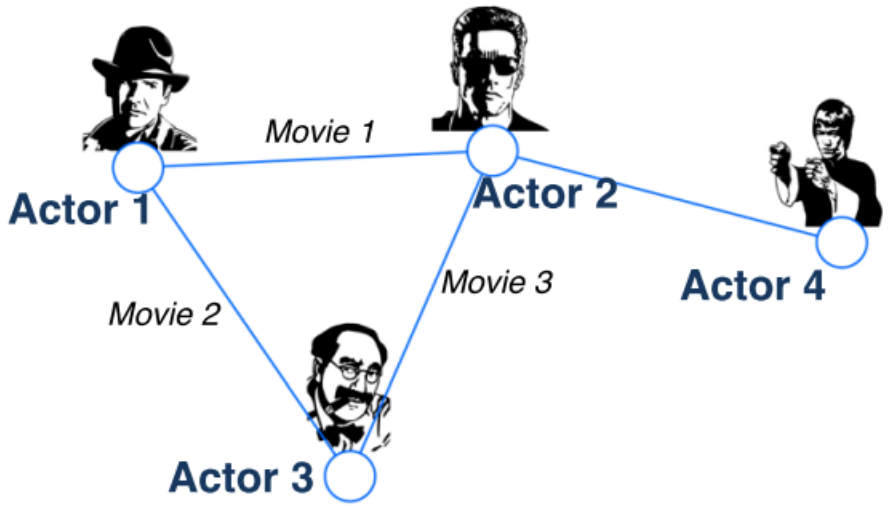
Graphs: A Universal Language

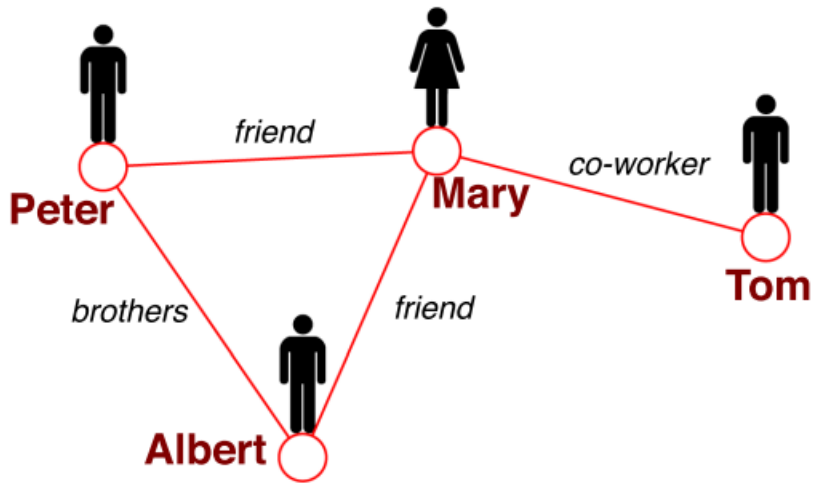
- Graphs are a general language for describing and modeling complex systems

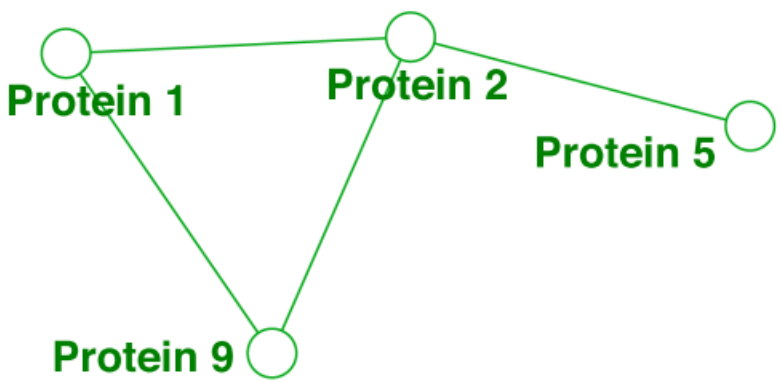


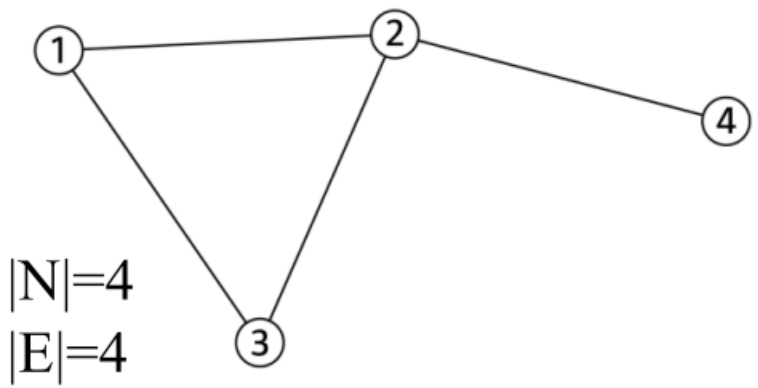
Graph!

Graphs: Common Language

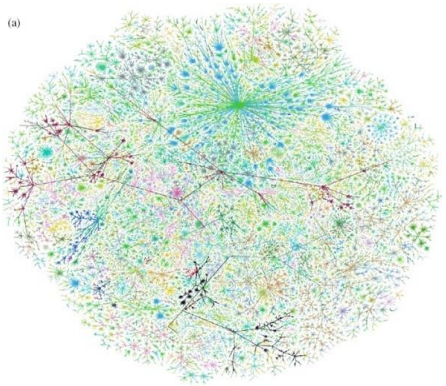






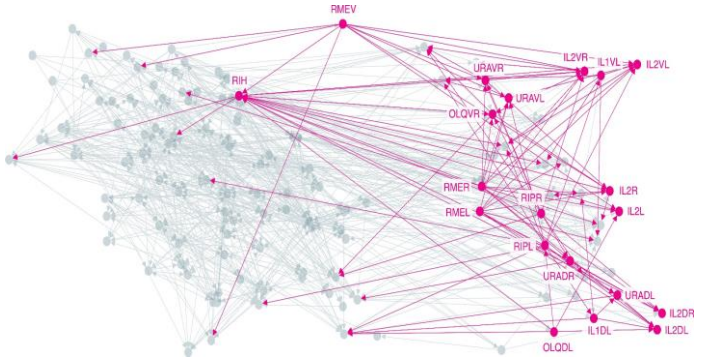


Graph-structured Data Are Ubiquitous

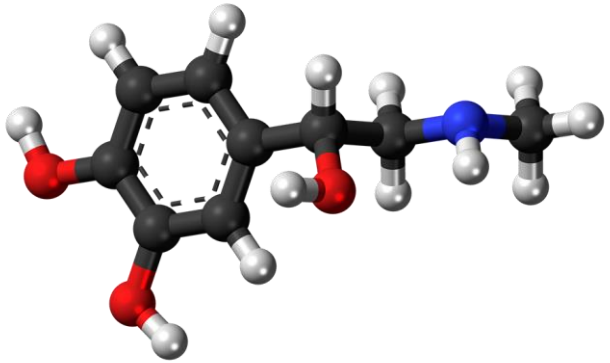


Internet

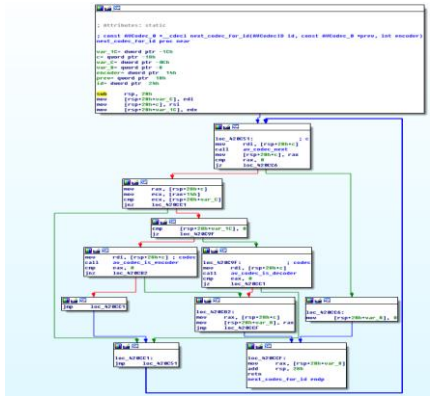
Social networks



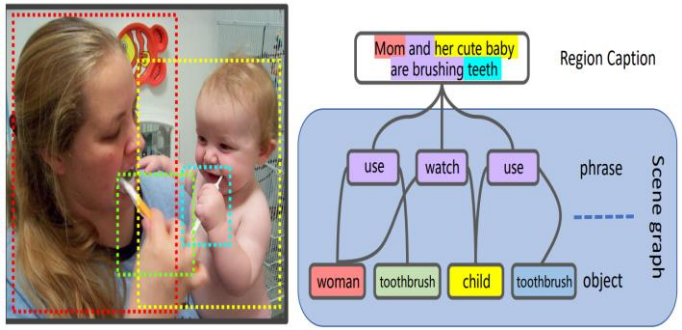
Networks of neurons



Biomedical graphs

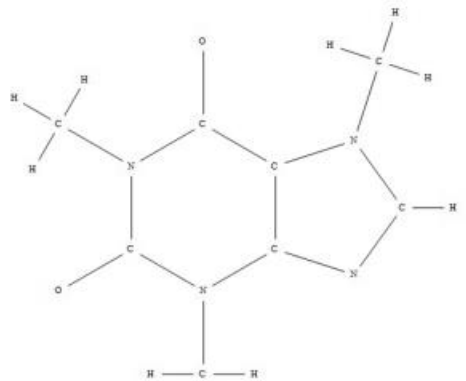


Program graphs

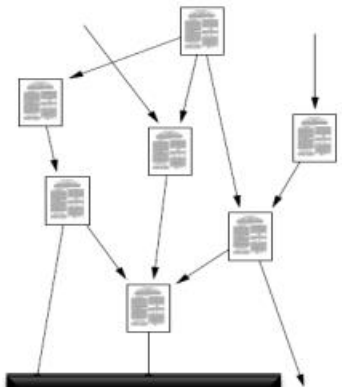


Scene graphs

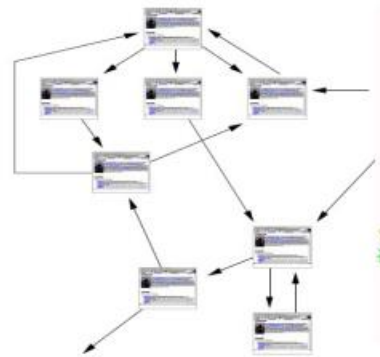
Networks around us!



Molecules

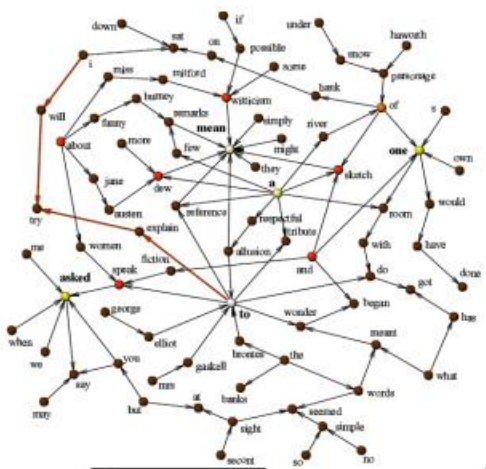


Knowledge

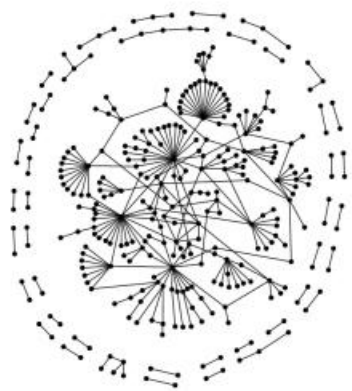


Information

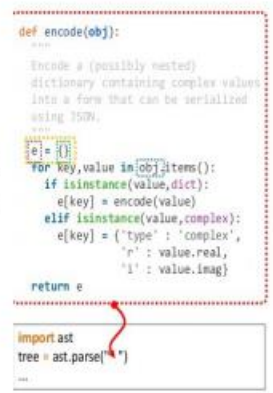
Brain/neurons

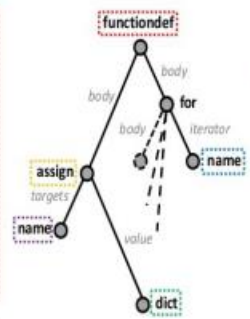


Genes



Communication





Software

Social

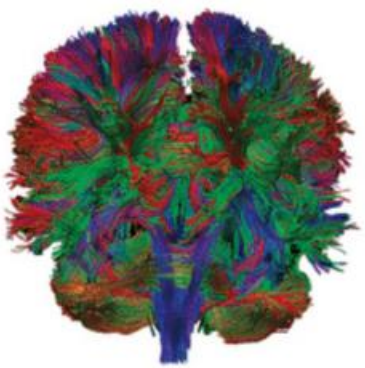
Applications of DL on Graphs

Computer graphics

Virtual/augmented reality

Robotics

Autonomous driving



Medicine

Drug design

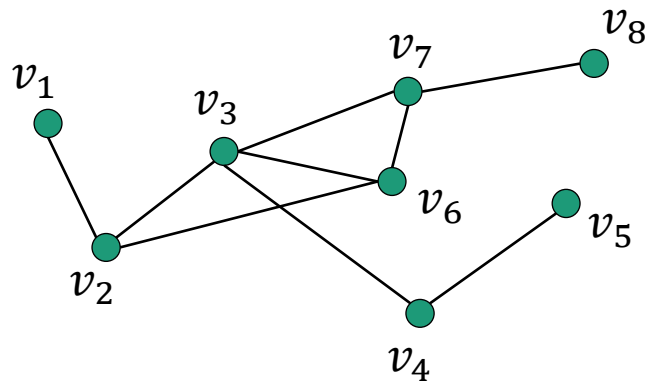
Why Graphs? Why Now?

- Universal language for describing complex data
  - Networks/graphs from science, nature, and technology are more similar than one would expect
- Shared vocabulary between fields
  - Computer Science, Social science, Physics, Biology, Economics
- Data availability (+ computational challenges)
  - Social/internet, text, logic, program, bio, health, and medical
- Impact
  - Social networking, Social media, Drug design, Event detection, Natural language processing, Computer vision, and Logic reasoning

Homogeneous vs Multi-relational vs Heterogeneous Graphs

Graph types	Homogeneous	Multi-relational	Heterogeneous
# of node types	1	1	> 1
# of edge types	1	> 1	>= 1

Graphs and Graph Signals

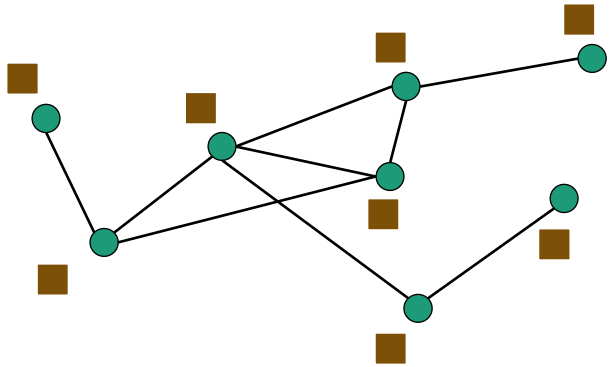


$$\mathcal{V} = \{v_1, \ldots, v_N\}$$

$$\mathcal{E} = \{e_1, \ldots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

Graphs and Graph Signals



Graph Signal: $f : \mathcal{V} \rightarrow \mathbb{R}^N$

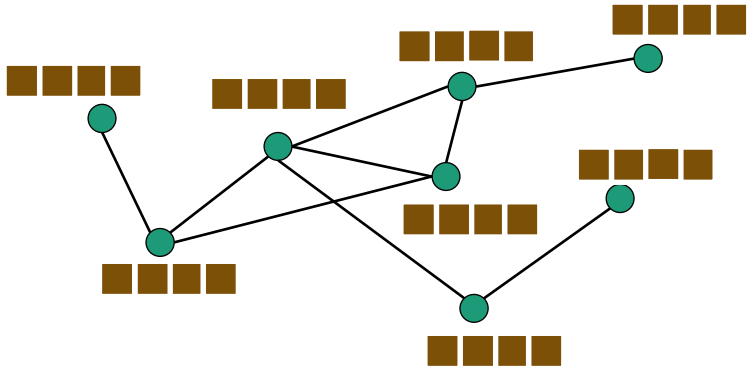
$$\mathcal{V} = \{v_1, \ldots, v_N\}$$

$$\mathcal{E} = \{e_1, \ldots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

$$\mathcal{V} \longrightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

Graphs and Graph Signals



Graph Signal: $f : \mathcal{V} \rightarrow \mathbb{R}^{N \times d}$

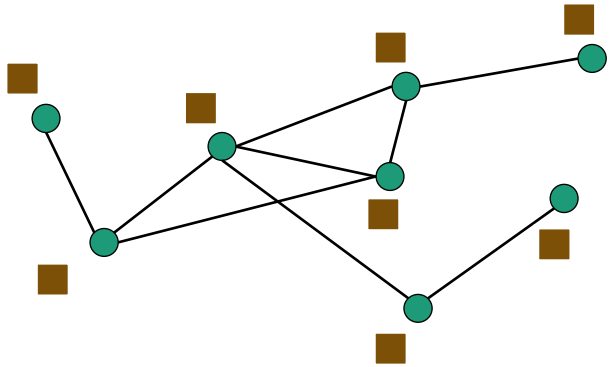
$$\mathcal{V} = \{v_1, \ldots, v_N\}$$

$$\mathcal{E} = \{e_1, \ldots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

$$\mathcal{V} \longrightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

Graphs and Graph Signals



Graph Signal: $f : \mathcal{V} \rightarrow \mathbb{R}^N$

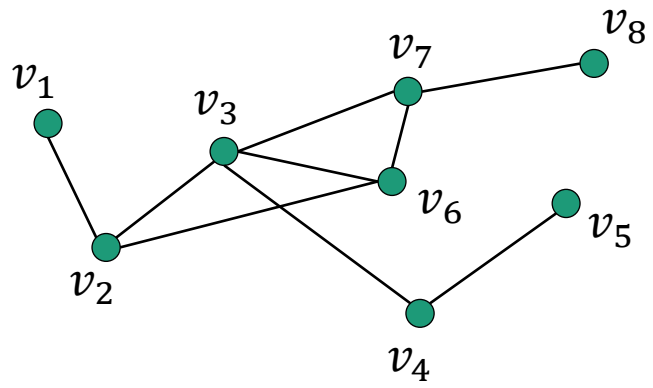
$$\mathcal{V} = \{v_1, \ldots, v_N\}$$

$$\mathcal{E} = \{e_1, \ldots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

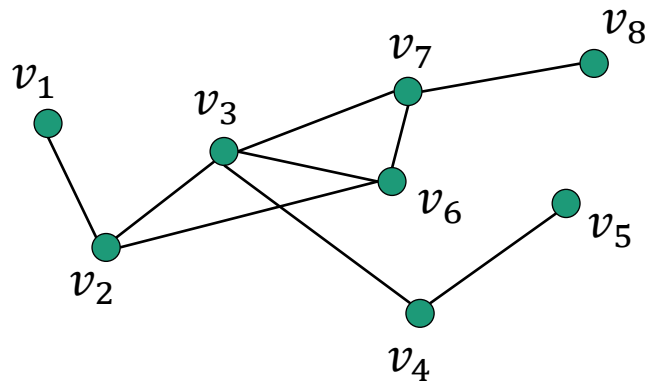
$$\mathcal{V} \longrightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

Matrix Representations of Graphs



Spectral graph theory. American Mathematical Soc.; 1997.

Matrix Representations of Graphs



Adjacency Matrix: $A[i,j] = 1$ if v_i is adjacent to v_j
$A[i,j] = 0$, otherwise

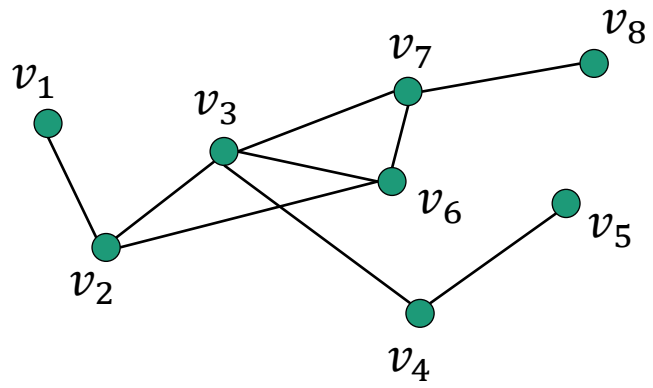
Adjacency Matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$\boldsymbol{A}$

Spectral graph theory. American Mathematical Soc.; 1997.

Matrix Representations of Graphs



Adjacency Matrix: $A[i,j] = 1$ if v_i is adjacent to v_j

$A[i,j] = 0$, otherwise

Degree Matrix: $\mathbf{D} = \mathrm{diag}(degree(v_1), \ldots, degree(v_N))$

Degree Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$\boldsymbol{D}$

$-$

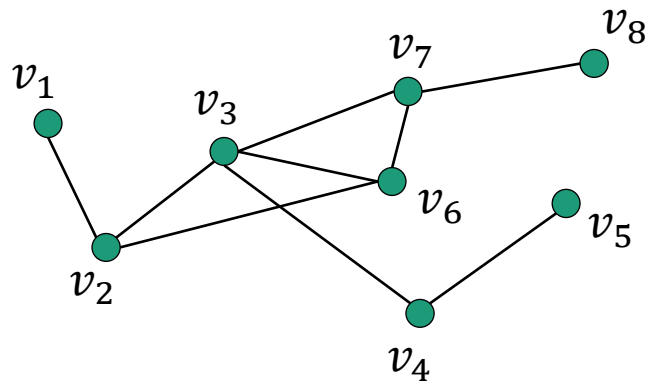
Adjacency Matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$\boldsymbol{A}$

Spectral graph theory. American Mathematical Soc.; 1997.

Matrix Representations of Graphs



Adjacency Matrix: $A[i,j] = 1$ if v_i is adjacent to v_j

$A[i,j] = 0$, otherwise

Degree Matrix: $\mathbf{D} = \mathrm{diag}(degree(v_1), \ldots, degree(v_N))$

Degree Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$\boldsymbol{D}$

$-$

Adjacency Matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$\boldsymbol{A}$

$=$

Laplacian Matrix

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

$\boldsymbol{L}$

Spectral graph theory. American Mathematical Soc.; 1997.

Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

Laplacian quadratic form:

$$\mathbf{f}^T \mathbf{L}\mathbf{f} = \frac{1}{2} \sum_{i,j=1}^{N} \mathbf{A}[i,j](\mathbf{f}(i) - \mathbf{f}(j))^2$$

Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

Laplacian quadratic form:

$$\mathbf{f}^T \mathbf{L}\mathbf{f} = \frac{1}{2} \sum_{i,j=1}^{N} \mathbf{A}[i,j](\mathbf{f}(i) - \mathbf{f}(j))^2$$

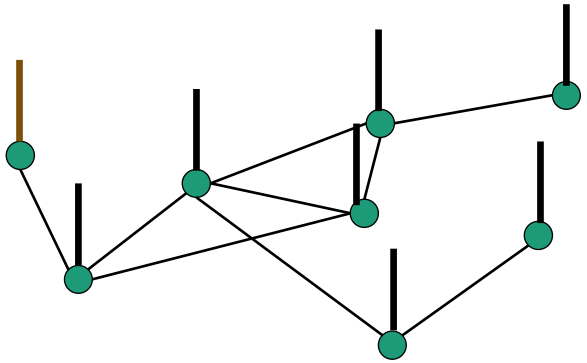
"Smoothness" or "Frequency" of the signal *f*

Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$



Low frequency graph signal

Laplacian quadratic form:

$$\mathbf{f}^T \mathbf{L}\mathbf{f} = \frac{1}{2} \sum_{i,j=1}^{N} \mathbf{A}[i,j](\mathbf{f}(i) - \mathbf{f}(j))^2$$

"Smoothness" or "Frequency" of the signal *f*

Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

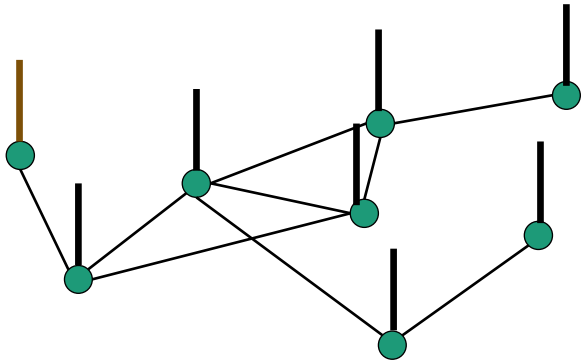
$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

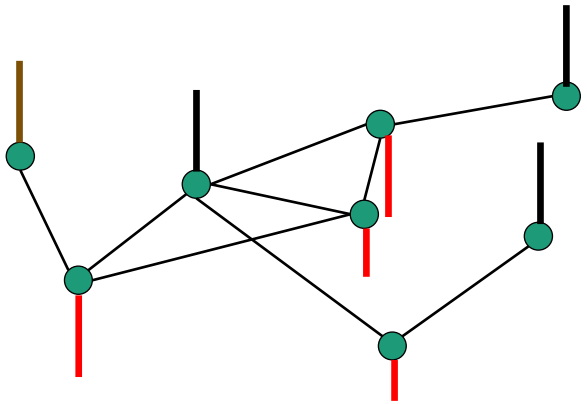
Laplacian quadratic form:

$$\mathbf{f}^T \mathbf{L}\mathbf{f} = \frac{1}{2} \sum_{i,j=1}^{N} \mathbf{A}[i,j](\mathbf{f}(i) - \mathbf{f}(j))^2$$

"Smoothness" or "Frequency" of the signal *f*



Low frequency graph signal



High frequency graph signal

Eigen-decomposition of Laplacian Matrix

Laplacian matrix has a complete set of orthonormal eigenvectors:

$$\mathbf{L} = \begin{bmatrix} | & & | \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \\ | & & | \end{bmatrix} \begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix} \begin{bmatrix} — & \mathbf{u}_0 & — \\ & \vdots & \\ — & \mathbf{u}_{N-1} & — \end{bmatrix}$$

Eigen-decomposition of Laplacian Matrix

Laplacian matrix has a complete set of orthonormal eigenvectors:

$$\mathbf{L} = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \\ | & & | \end{bmatrix}}_{\boldsymbol{U}} \underbrace{\begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix}}_{\boldsymbol{\Lambda}} \underbrace{\begin{bmatrix} — & \mathbf{u}_0 & — \\ & \vdots & \\ — & \mathbf{u}_{N-1} & — \end{bmatrix}}_{\boldsymbol{U^T}}$$

Eigen-decomposition of Laplacian Matrix

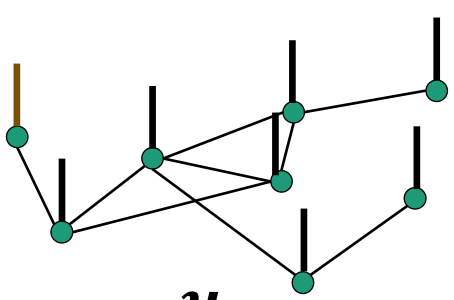
Laplacian matrix has a complete set of orthonormal eigenvectors:

$$\mathbf{L} = \underbrace{\begin{bmatrix} | & & | \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \\ | & & | \end{bmatrix}}_{\boldsymbol{U}} \underbrace{\begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix}}_{\boldsymbol{\Lambda}} \underbrace{\begin{bmatrix} — & \mathbf{u}_0 & — \\ & \vdots & \\ — & \mathbf{u}_{N-1} & — \end{bmatrix}}_{\boldsymbol{U^T}}$$

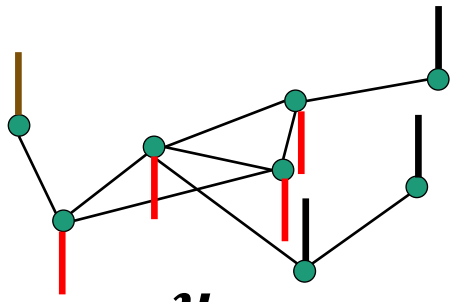
Eigenvalues are sorted non-decreasingly:

$$0 = \lambda_0 < \lambda_1 \leq \cdots \lambda_{N-1}$$

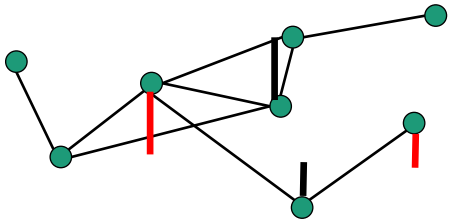
Eigenvectors as Graph Signals



$\boldsymbol{u}_0$



$\boldsymbol{u}_1$



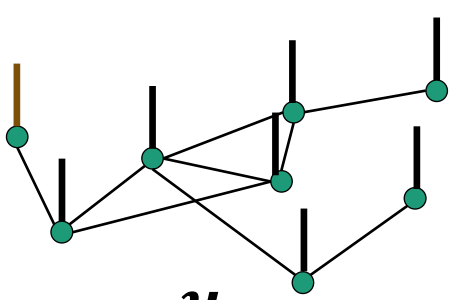
$\boldsymbol{u}_8$

Eigenvectors as Graph Signals

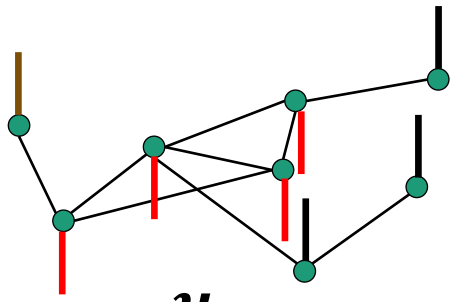
The frequency of an eigenvector of Laplacian matrix is its corresponding eigenvalue:

$$\mathbf{u}_i^T \mathbf{L} \mathbf{u}_i = \mathbf{u}_i^T \lambda_i \mathbf{u}_i = \lambda_i$$

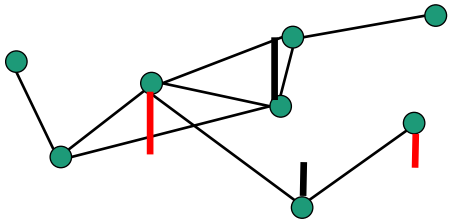
Frequency of the signal $\boldsymbol{u}_i$



$\boldsymbol{u}_0$



$\boldsymbol{u}_1$



$\boldsymbol{u}_8$

$\boldsymbol{u_0^T L u_0 = \lambda_0 = 0}$

$\boldsymbol{u_1^T L u_1 = \lambda_1}$

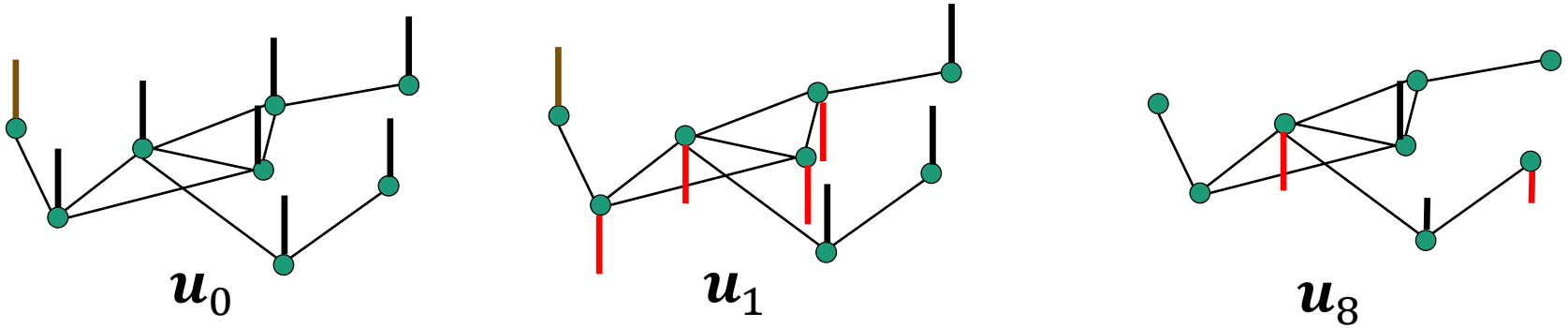
$\boldsymbol{u_8^T L u_8 = \lambda_8}$

Eigenvectors as Graph Signals

The frequency of an eigenvector of Laplacian matrix is its corresponding eigenvalue:

$$\mathbf{u}_i^T \mathbf{L} \mathbf{u}_i = \mathbf{u}_i^T \lambda_i \mathbf{u}_i = \lambda_i$$

Frequency of the signal $\boldsymbol{u}_i$



$\boldsymbol{u_0^T L u_0 = \lambda_0 = 0}$ $\boldsymbol{u_1^T L u_1 = \lambda_1}$ $\boldsymbol{u_8^T L u_8 = \lambda_8}$

Graph Fourier Transform

A signal f can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \hat{f}_i \cdot u_i$$

u_i: graph Fourier mode λ_i: frequency $\hat{f}_i$: graph Fourier coefficients

Graph Fourier Transform

A signal f can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \underbrace{\hat{f}_i}_{f^T u_i} \cdot u_i$$

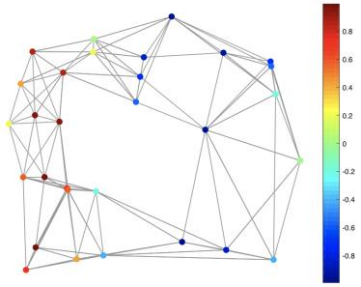
u_i: graph Fourier mode λ_i: frequency $\hat{f}_i$: graph Fourier coefficients

Graph Fourier Transform (GFT)

A signal f can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \underbrace{\hat{f}_i}_{f^T u_i} \cdot u_i$$

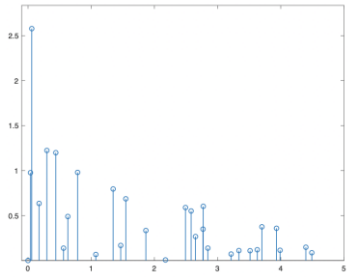
u_i: graph Fourier mode λ_i: frequency $\hat{f}_i$: graph Fourier coefficients



Spatial domain: $\boldsymbol{f}$

$$\hat{\boldsymbol{f}} = \boldsymbol{U}^T \boldsymbol{f}$$

Decompose signal f



Spectral domain: $\hat{\boldsymbol{f}}$

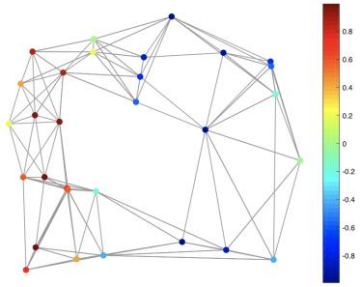
The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*

Inverse Graph Fourier Transform (IGFT)

A signal f can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \underbrace{\hat{f}_i}_{f^T u_i} \cdot u_i$$

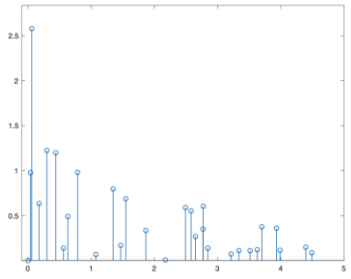
u_i: graph Fourier mode λ_i: frequency $\hat{f}_i$: graph Fourier coefficients



Spatial domain: $\boldsymbol{f}$

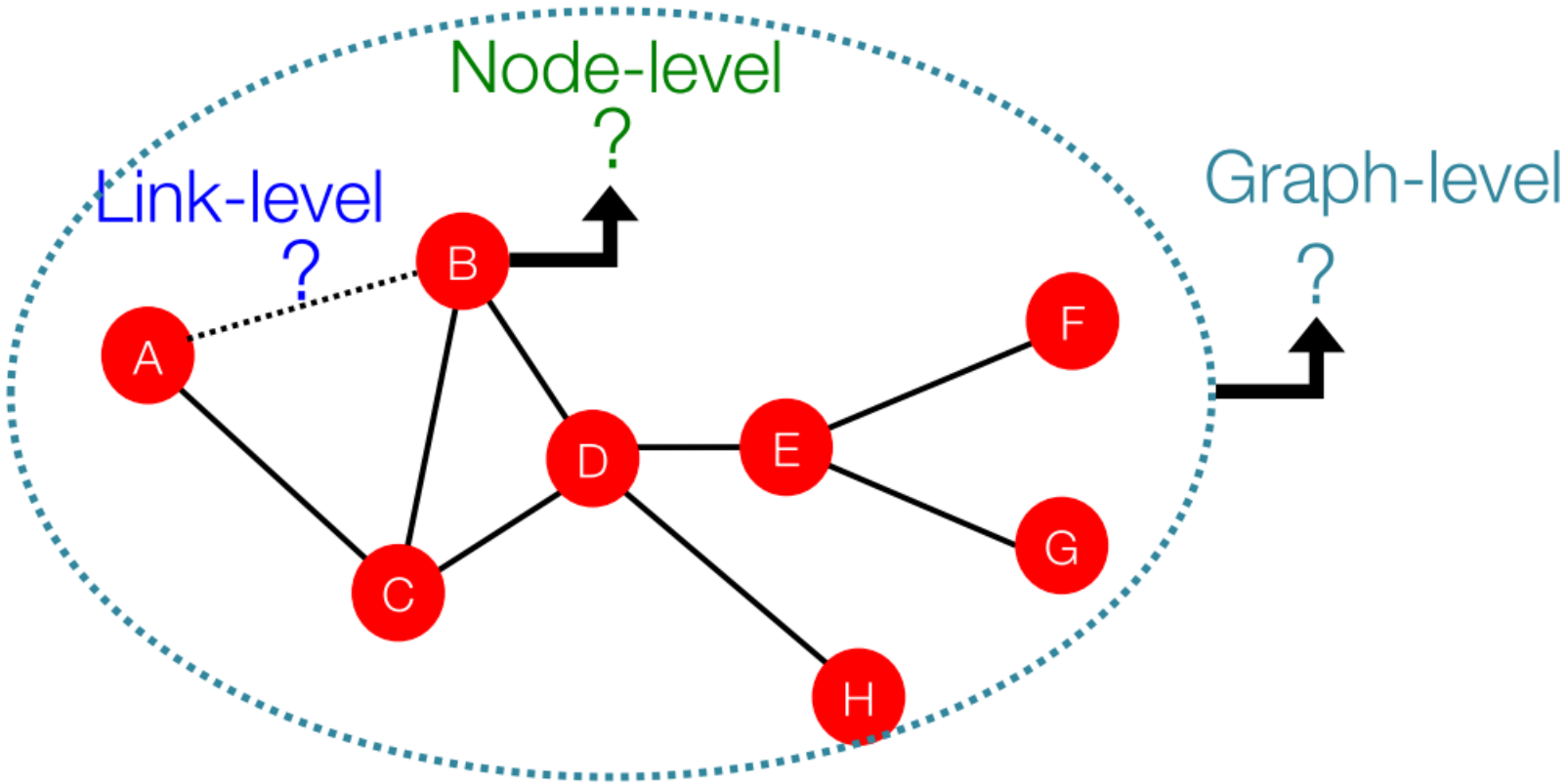
$\boldsymbol{f} = \boldsymbol{U}\hat{\boldsymbol{f}}$

Reconstruct signal $\boldsymbol{f}$

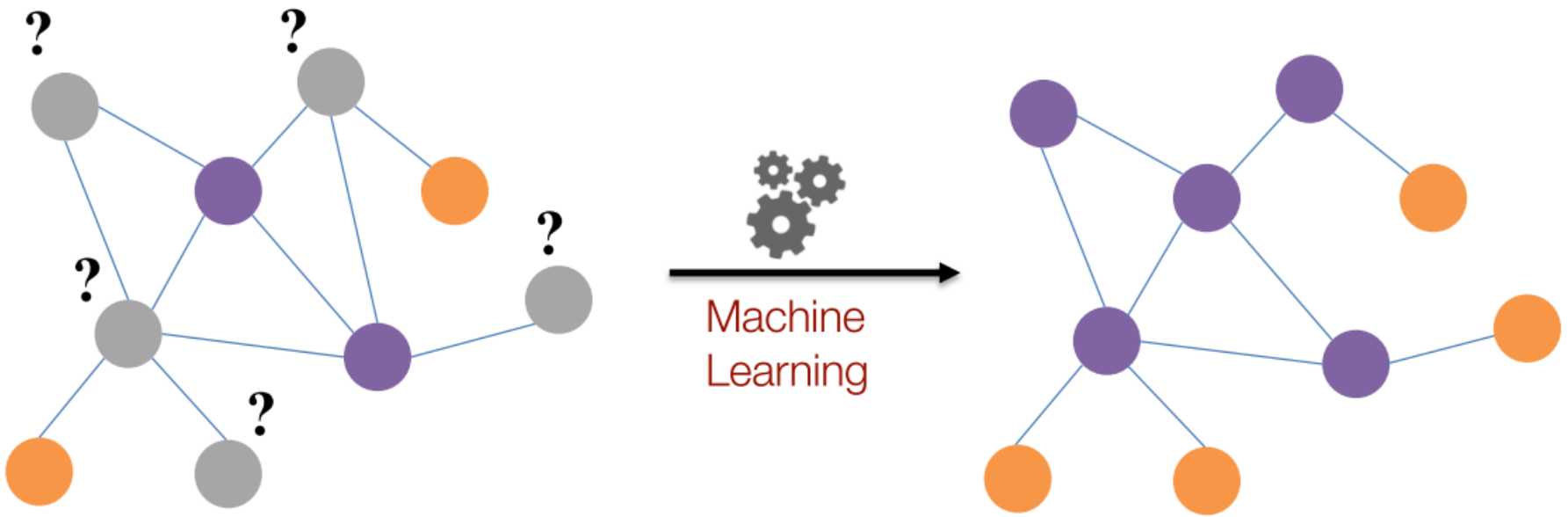


Spectral domain: $\hat{\boldsymbol{f}}$

Machine Learning Tasks

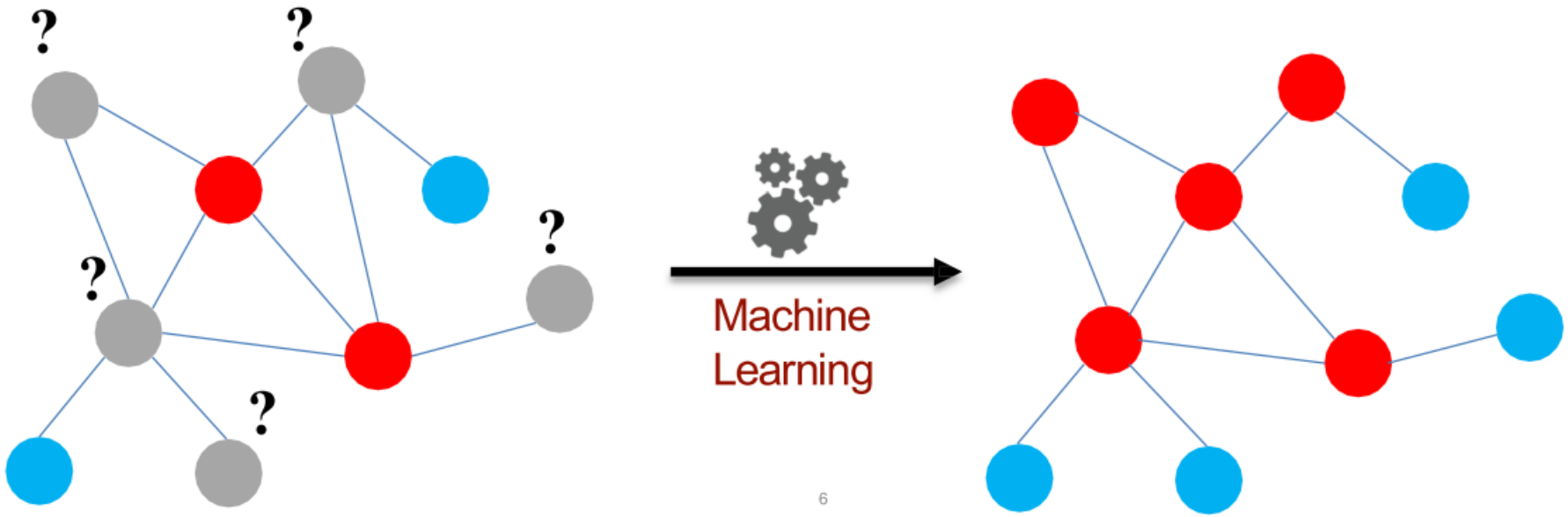


Example: Node Classification



- What users are going to churn?
- What is the disease of a patient?
- What are functions of proteins?

Node Classification Task



Node Classification: An Example

Classifying the function of proteins in the interactome!

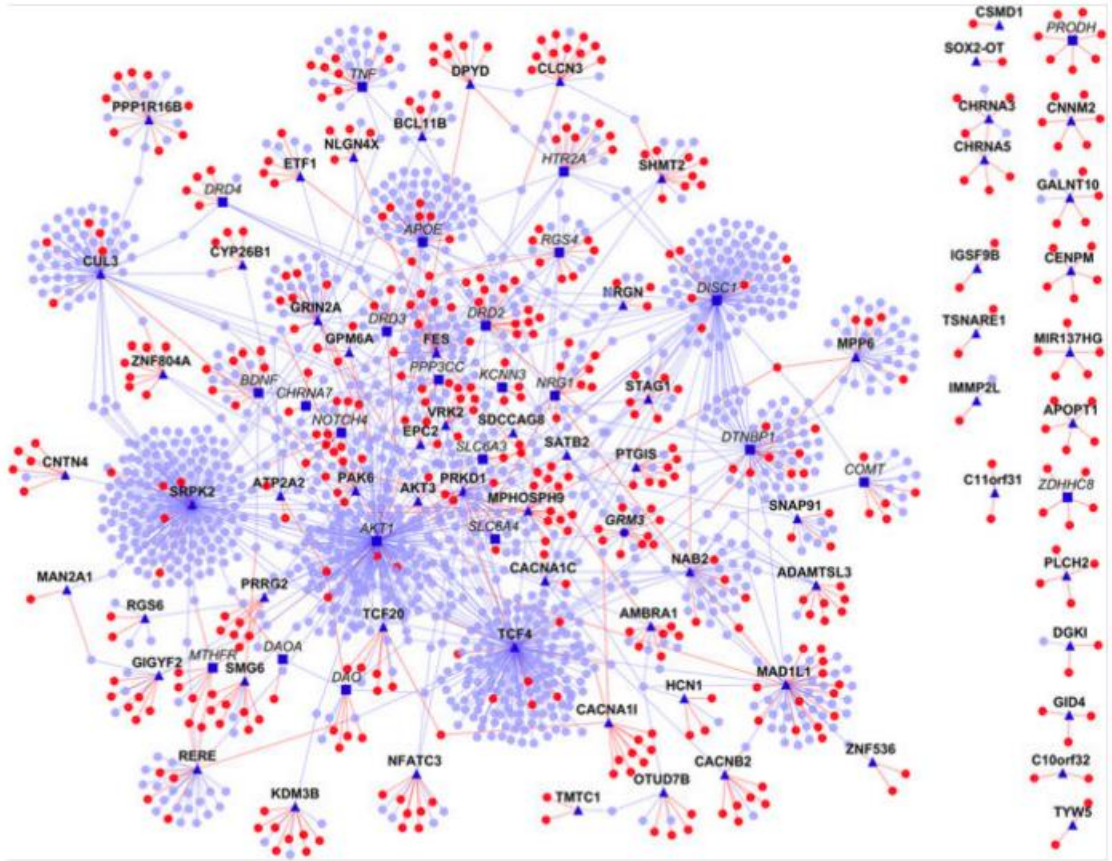
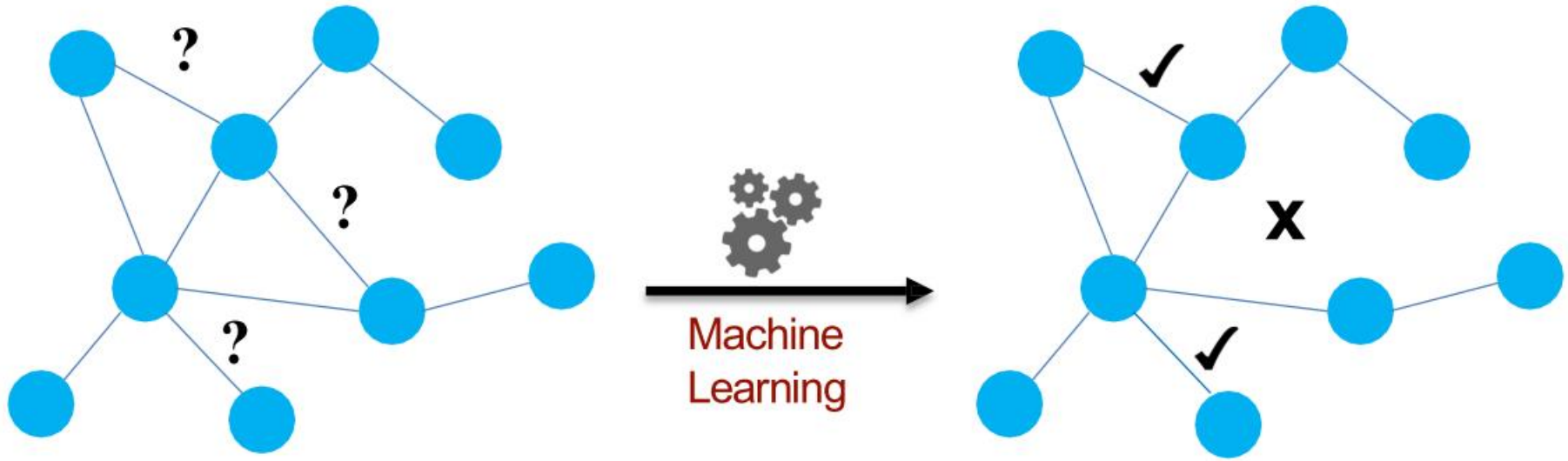


Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

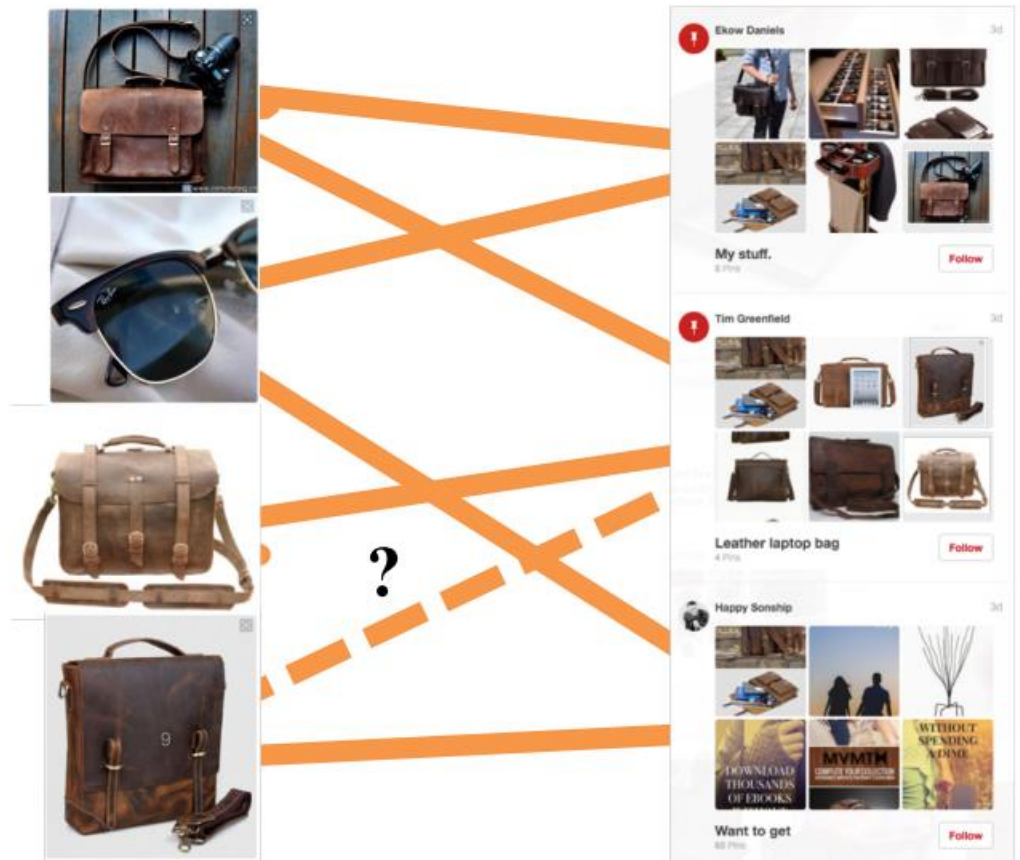
Link Prediction Task



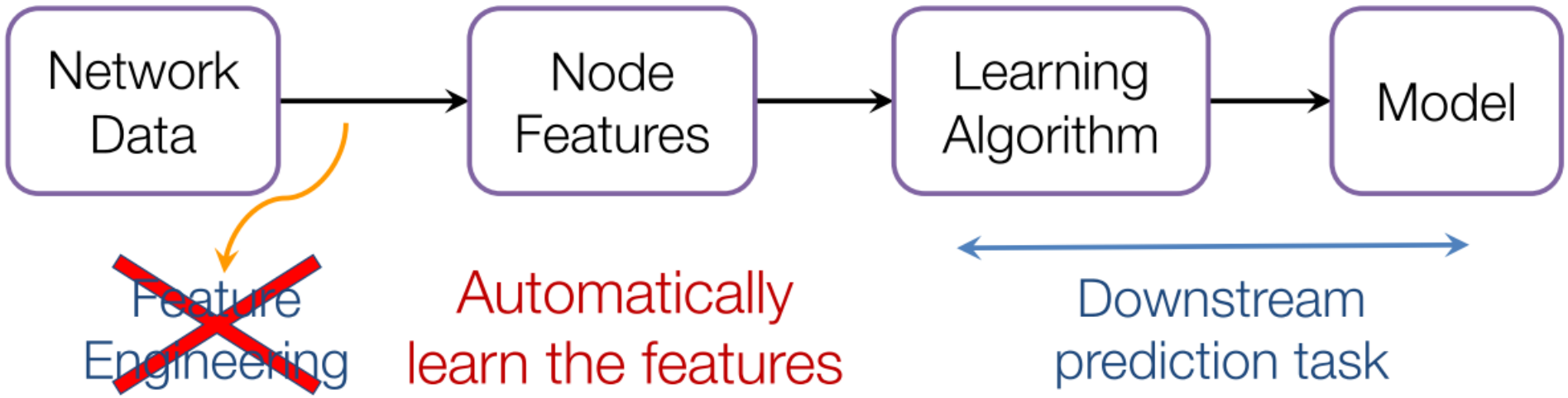
8

Link Prediction: An Example

Content recommendation is link prediction!



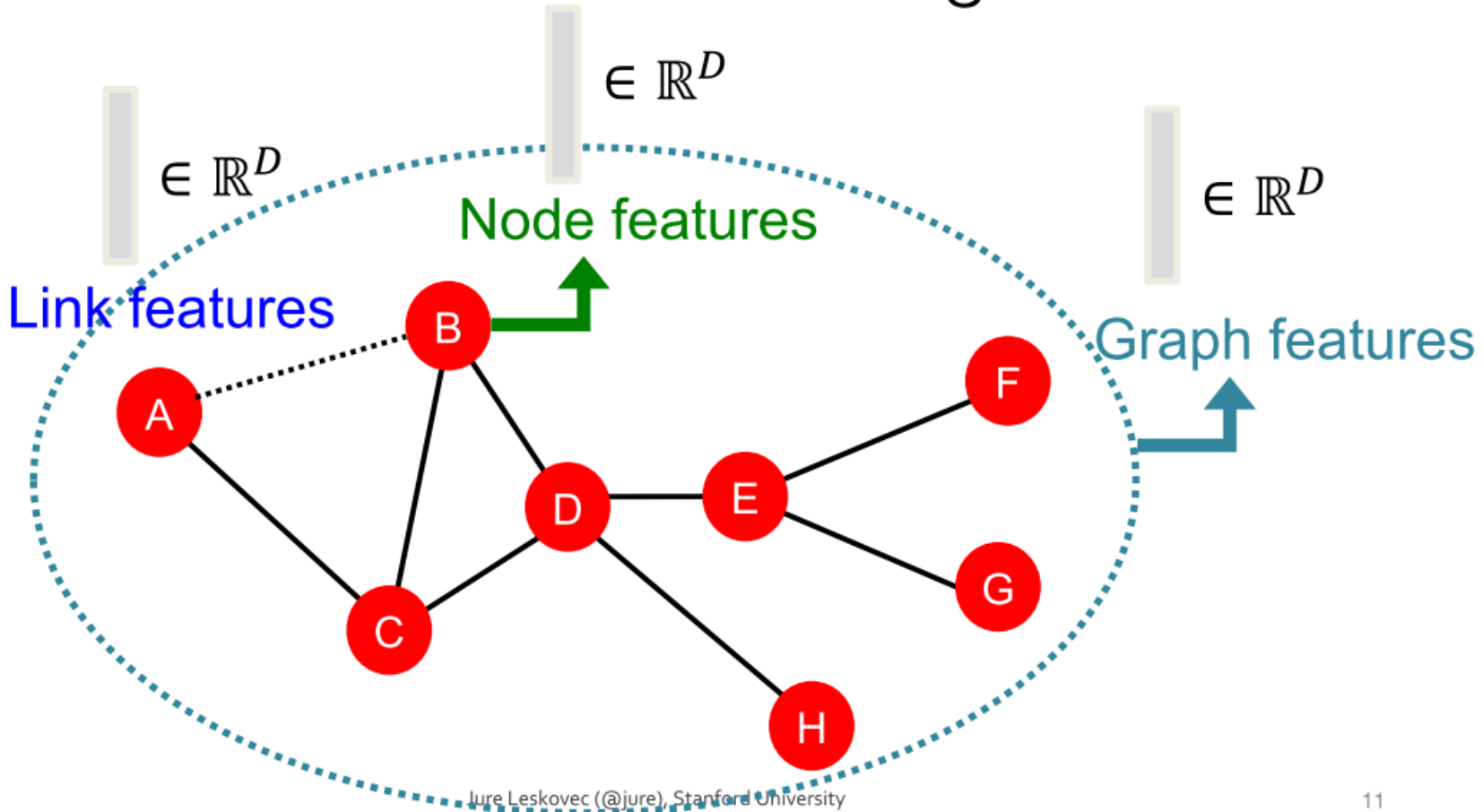
Machine Learning Lifecycle



(Supervised) Machine Learning Lifecycle:
This feature, that feature.
Every single time!

Graph Feature Engineering

- Design features for nodes/links/graphs
- Obtain features for all training data



Two Pain Points: One

Data Scientist's pain point #1:

- Data scientists have to hand encode features to solve prediction problems.
- Hand encoding graph features is…

… complex and involves expensive queries

… error prone

… suboptimal

… labor intensive

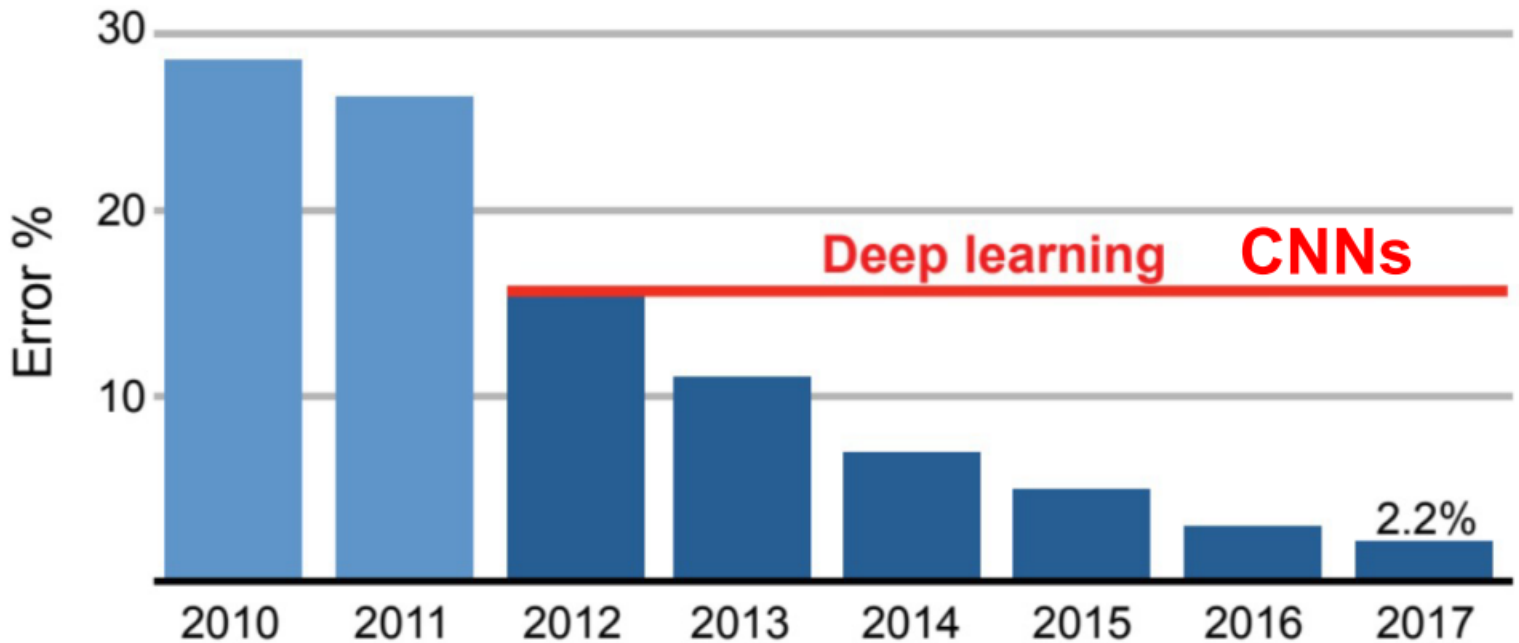
Two Pain Points: Two

Data Scientist's pain point #2:

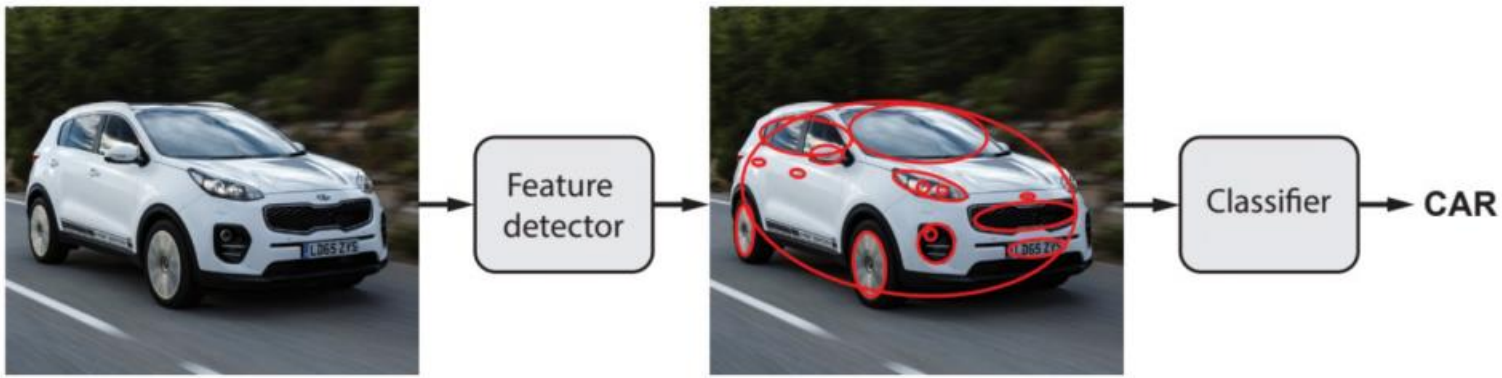
- Data is often incomplete.
  - Address Books, Follows, Interests, Protein Protein Interaction, Ancestry
- Entity information is incomplete.
- Predictions often entail completing the "missing information".
  - Relational structure is often not leveraged due to scalability issues.

The Deep Learning Revolution

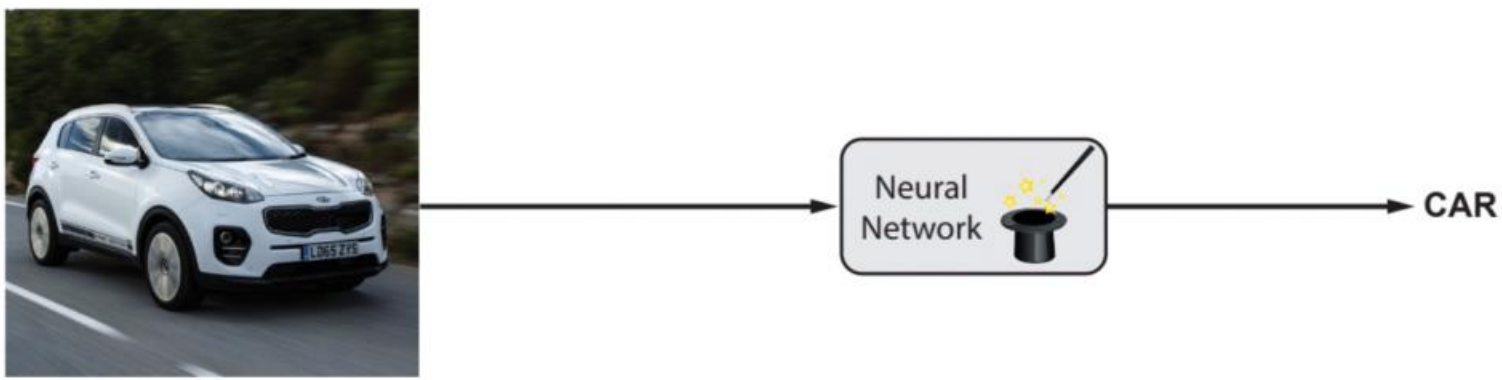
Breakthroughs in image recognition fueled by Convolutional Neural Networks.



Representation Learning



Classical computer vision: hand-crafted features (e.g. SIFT)
+ simple classifier (e.g. SVM)



Modern computer vision: data-driven end-to-end systems

Doubt thou the stars are fire,
Doubt that the sun doth move,
Doubt truth to be a liar,
But never doubt I love...

Text

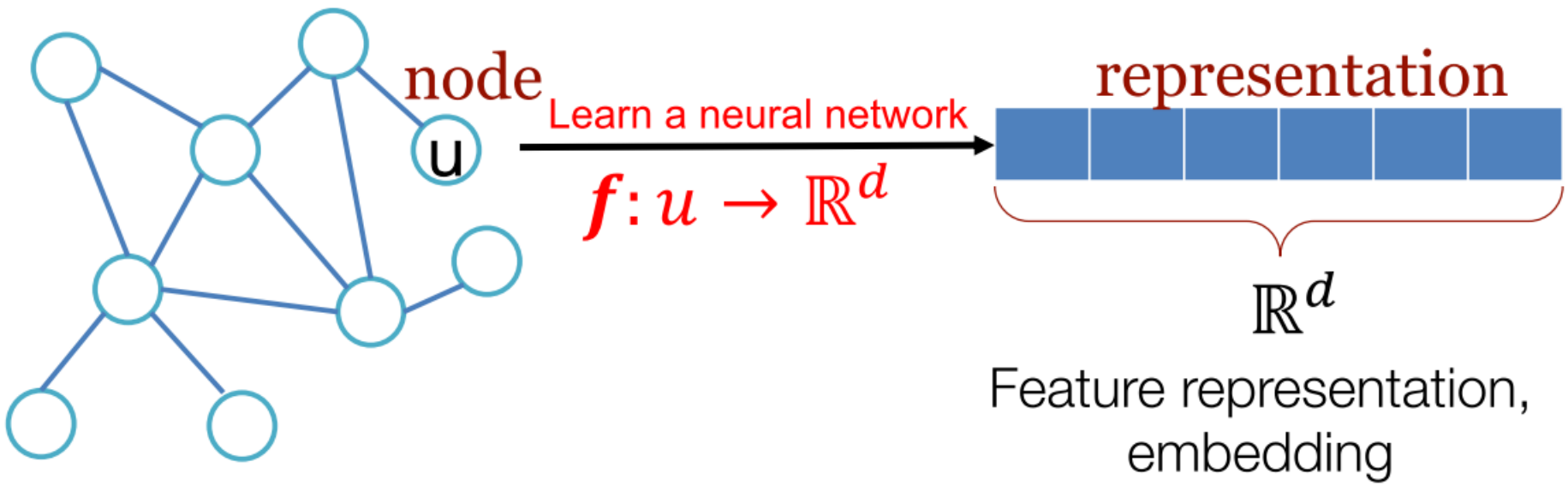
Audio signals

Images

But, modern deep learning toolbox is designed for sequences & grids

Goal: Representation Learning

Map nodes to d-dimensional **embeddings** such that similar nodes in the network are embedded close together

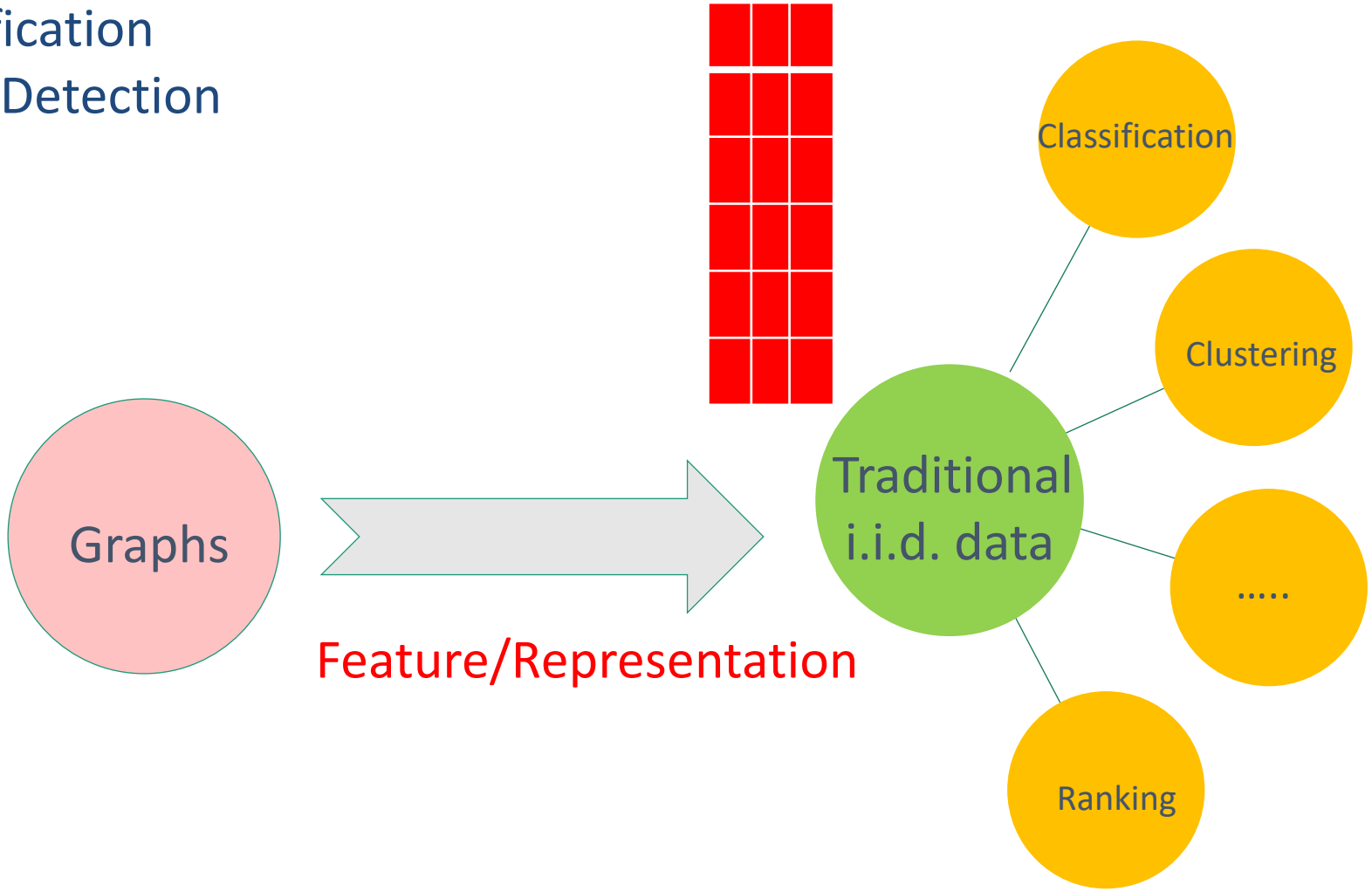


ML on Graphs

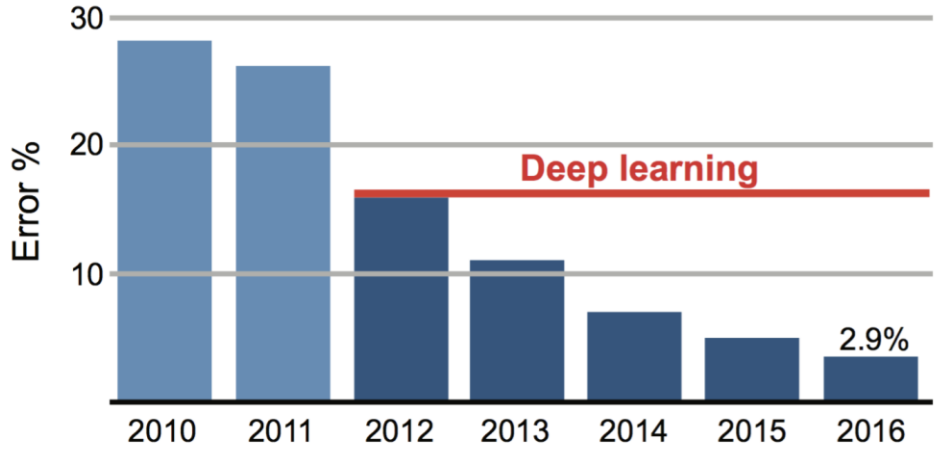
Numerous real-world problems can be summarized as a set of tasks on graphs

- Link prediction
- Node Classification
- Community Detection
- Ranking ….

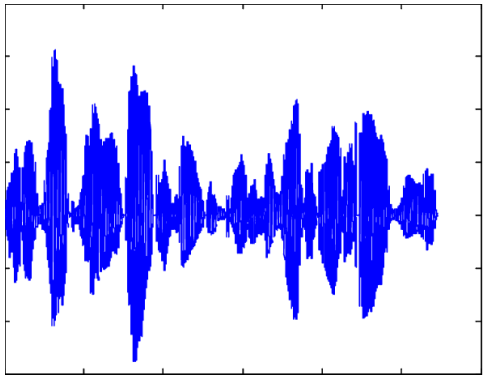
ML solutions



The Power of Deep Learning



IMAGENET



Acoustic model	Recog \ WER	RT03S FSH	Hub5 SWB
Traditional features	1-pass –adapt	27.4	23.6
Deep Learning	1-pass –adapt	18.5	16.1

Machine(Deep) Learning with Graphs

Classical ML tasks in graphs:

- Node classification
  - Predict a type of a given node
- Link prediction
  - Predict whether two nodes are linked
- Community detection
  - Identify densely linked clusters of nodes
- Graph similarity
  - How similar are two (sub)graphs

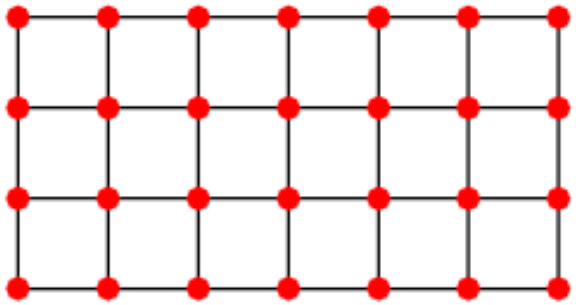
Recent ML tasks in graphs:

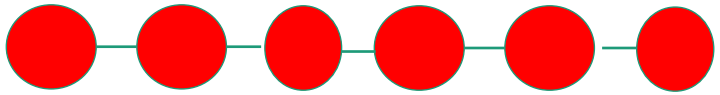
- Graph classification
  - Predict a type of a given graph
- Graph generation
  - Generate graphs from learned distribution
- Graph structure learning
  - Identify densely linked clusters of nodes
- Graph-to-XXX learning 
  - Graph Inputs – XXX outputs

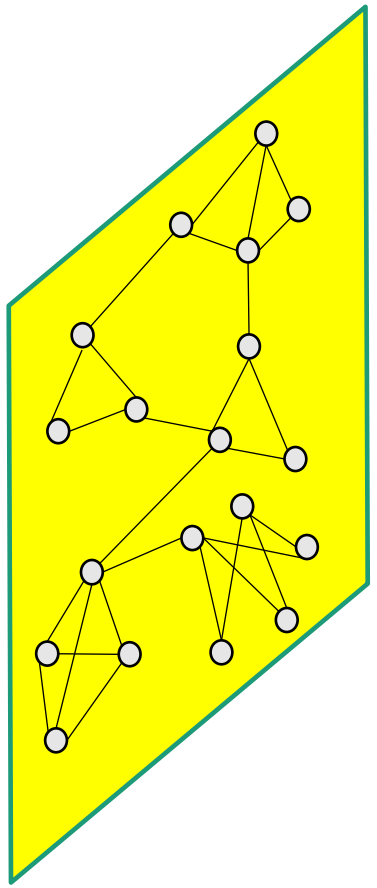
Deep Learning Meets Graphs: Challenges

Traditional DL is designed for simple grids or sequences

- CNNs for fixed-size images/grids
- RNNs for text/sequences



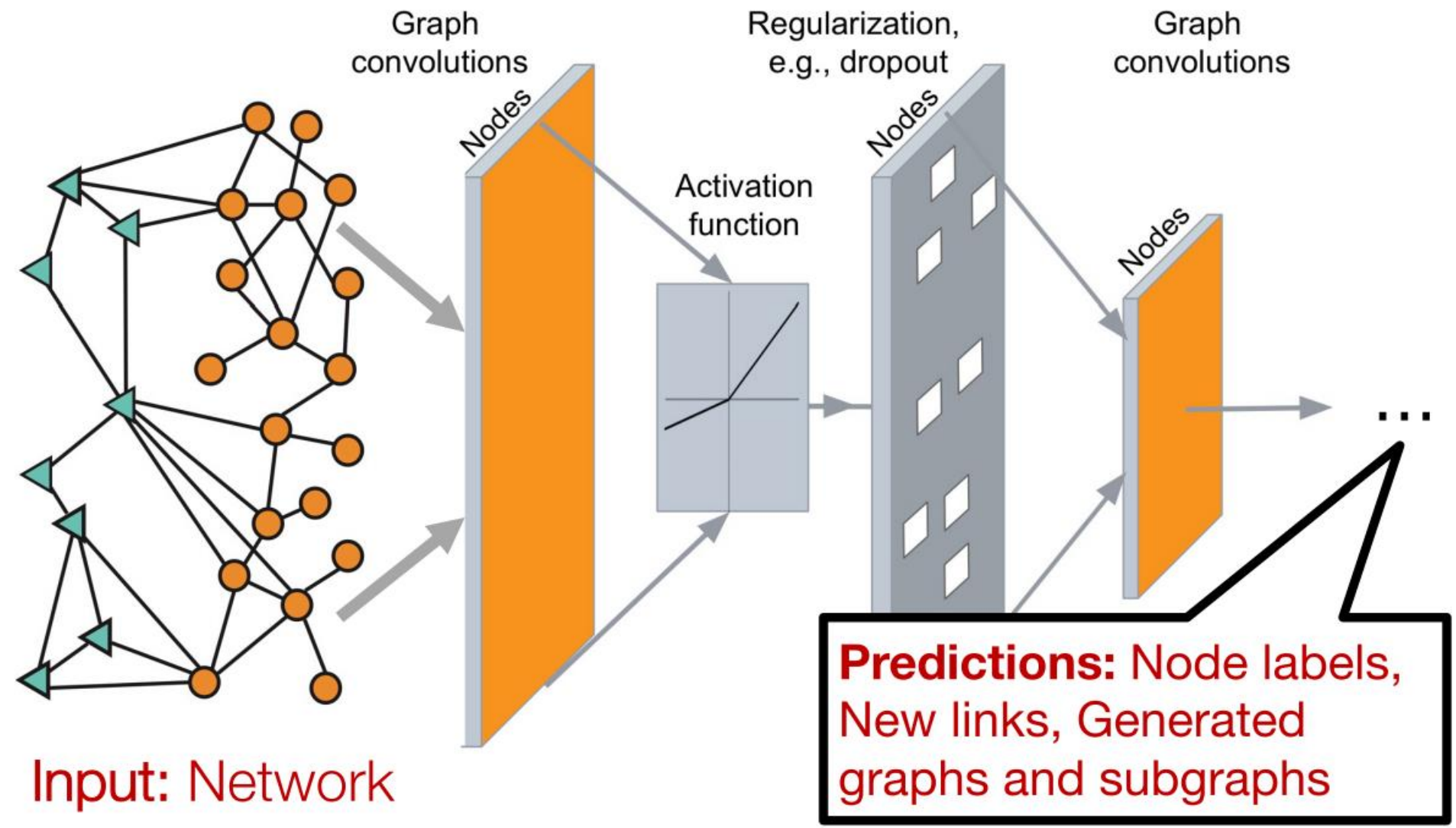




But nodes on graphs have different connections

- Arbitrary neighbor size
- Complex topological structure
- No fixed node ordering

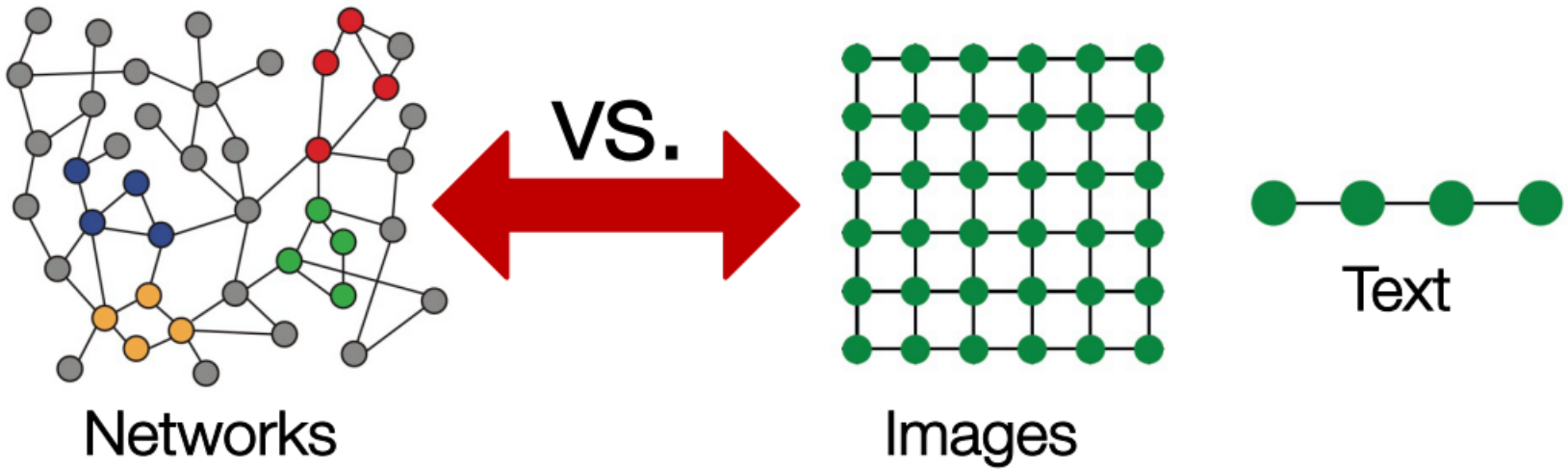
Deep Learning in Graphs



Why is it Hard?

Networks are complex!

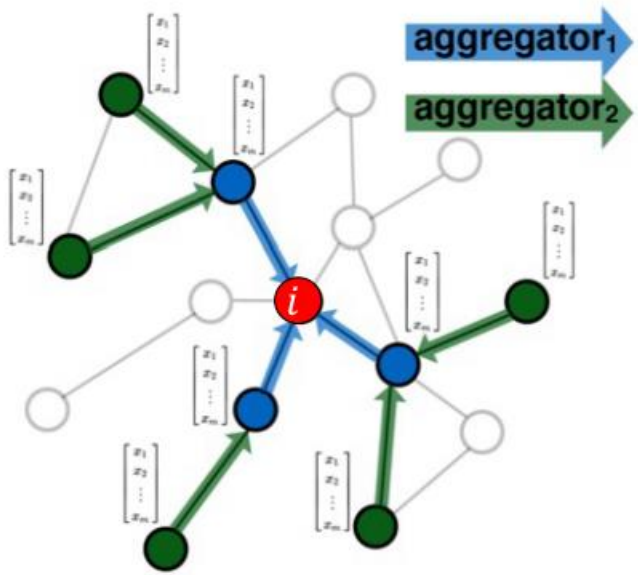
- Arbitrary size and complex topological structure (i.e., no spatial locality like grids)



- No fixed node ordering or reference point
- Often dynamic and have multimodal features

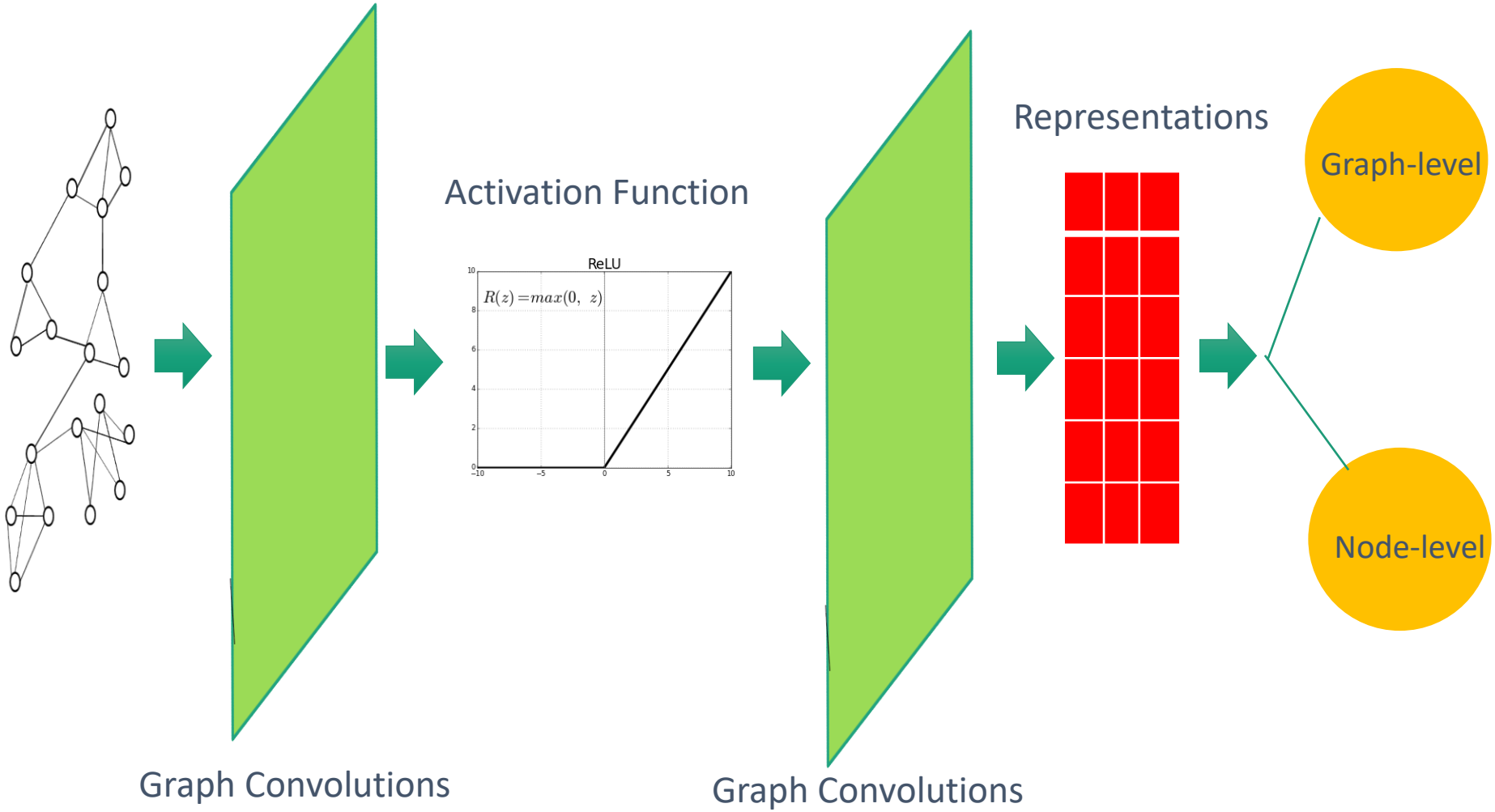
Networks as computation graphs

Key idea: Network is a computation graph



Learn how to propagate information across the network

Graph Neural Networks



Graph Representation Learning (GNNs)

- Graph Neural Networks (GNNs) extends the well known CNN and RNN on graphs, from Euclidean data to Graphs and Manifolds

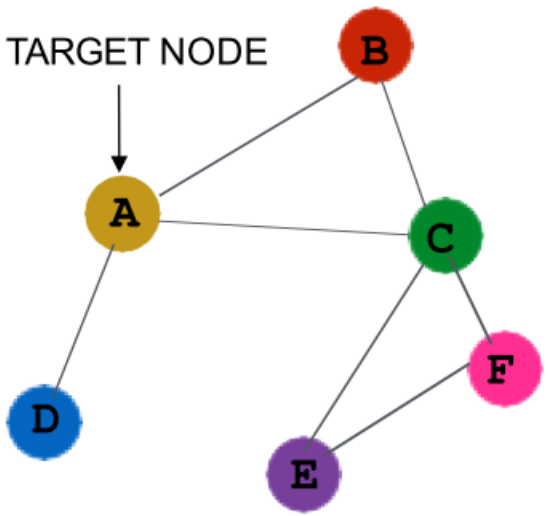
- RNN-based GNNs:
  - Graph neural networks (Scarselli et al., 2009)
  - Gated graph sequence neural networks (GGS-NNs) (Li et al., ICLR 2016)

- CNN-based GNNs:
  - Graph Convolutional Networks (GCN) (Kipf & Welling, ICLR 2017)

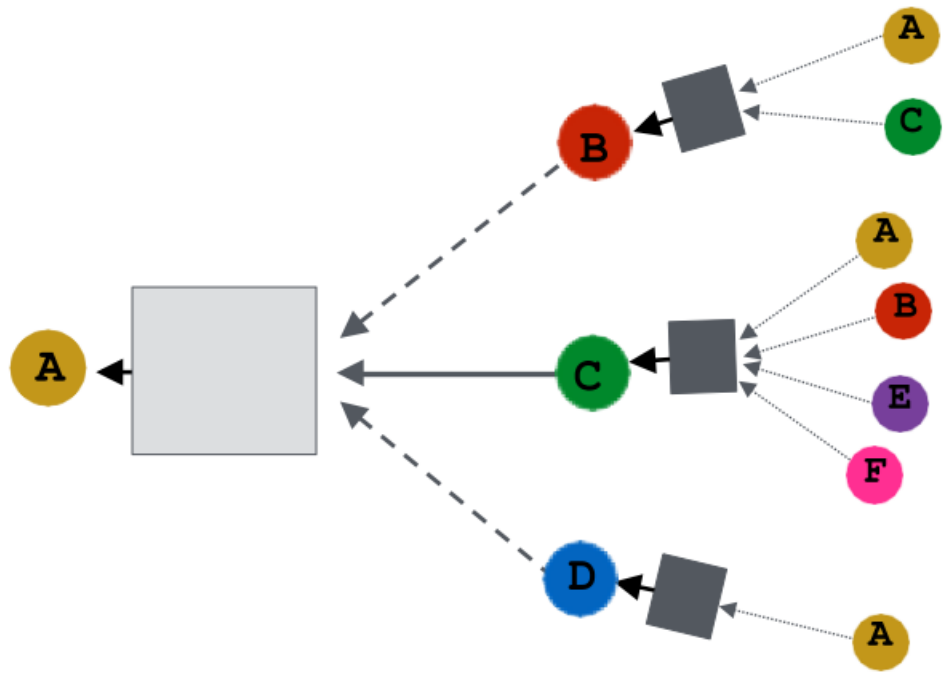
- Message Passing-based GNNs:
  - GraphSAGE (Hamilton & Ying & Leskovec, NIPS 2017)
  - Graph Attention Networks (GAT) (Velickovic et al., ICLR 2018)
  - MPNN (Gilmer et al., ICML 2017)

Graph Neural Networks

- Key idea: Generate node embeddings based on local neighborhoods.

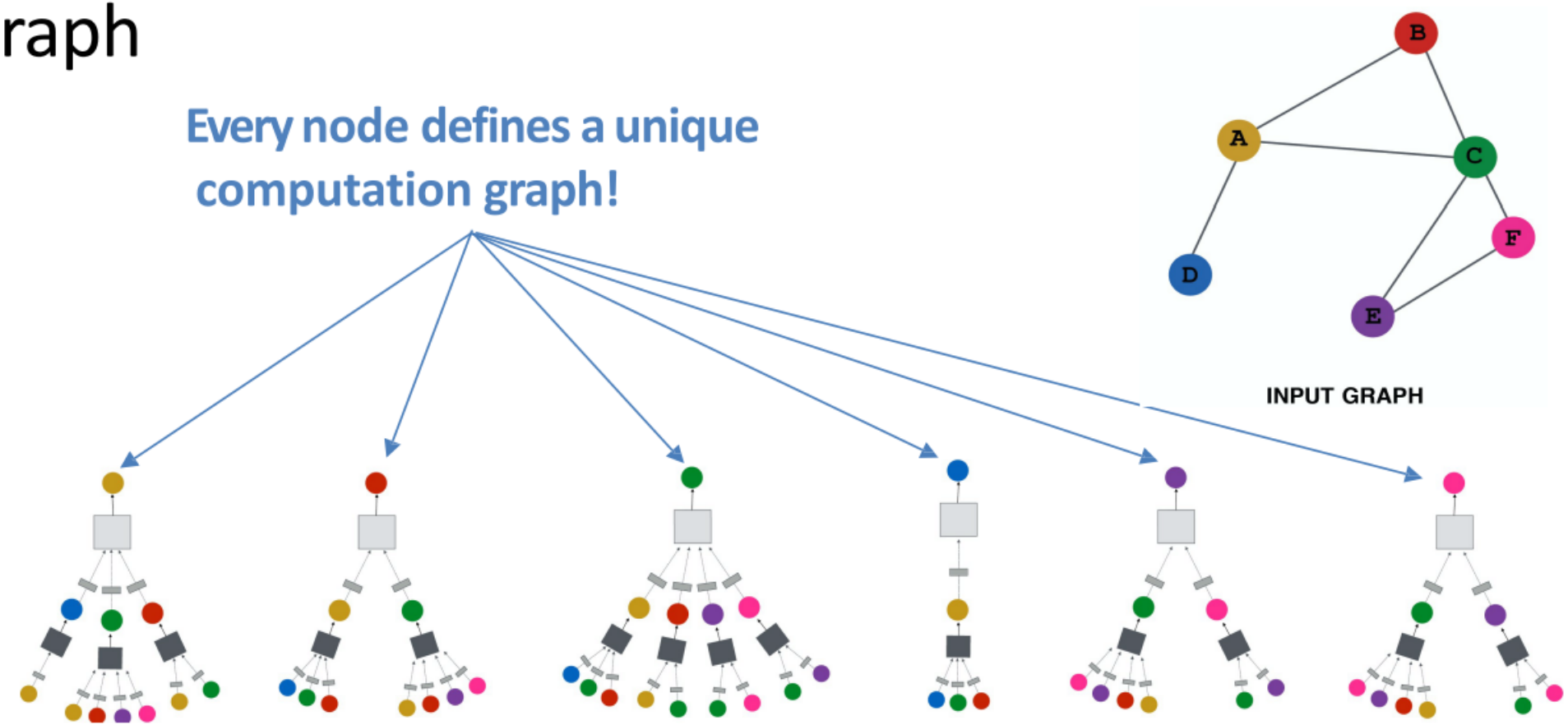


INPUT GRAPH



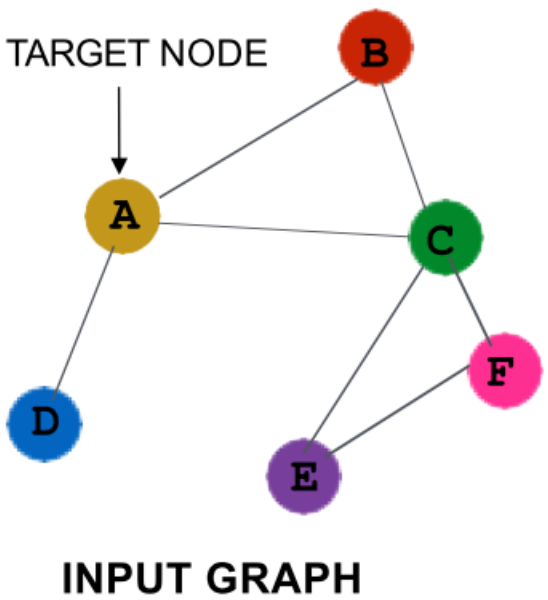
Neighborhood Aggregation

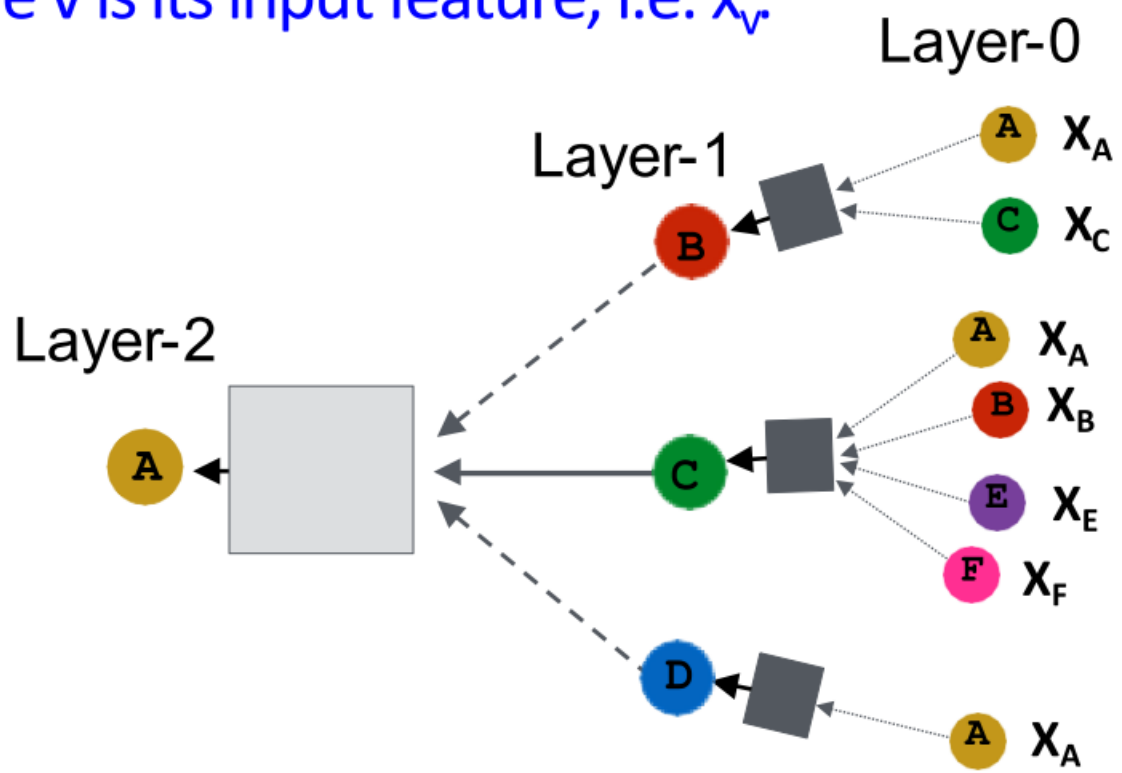
- Intuition: Network neighborhood defines a computation graph



Neighborhood Aggregation

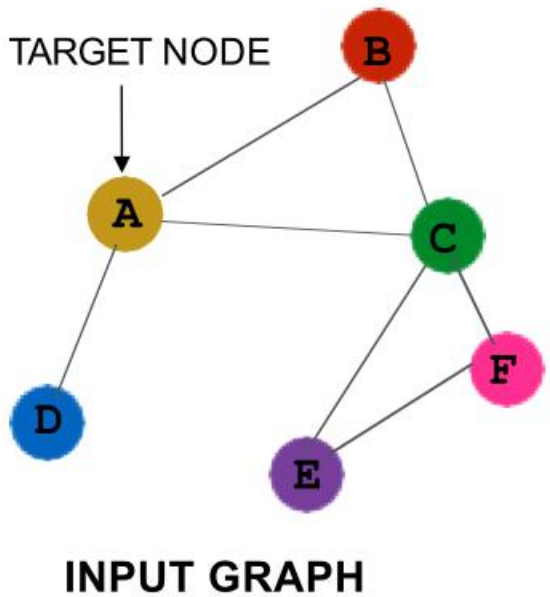
- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- "layer-0" embedding of node v is its input feature, i.e. x_v.



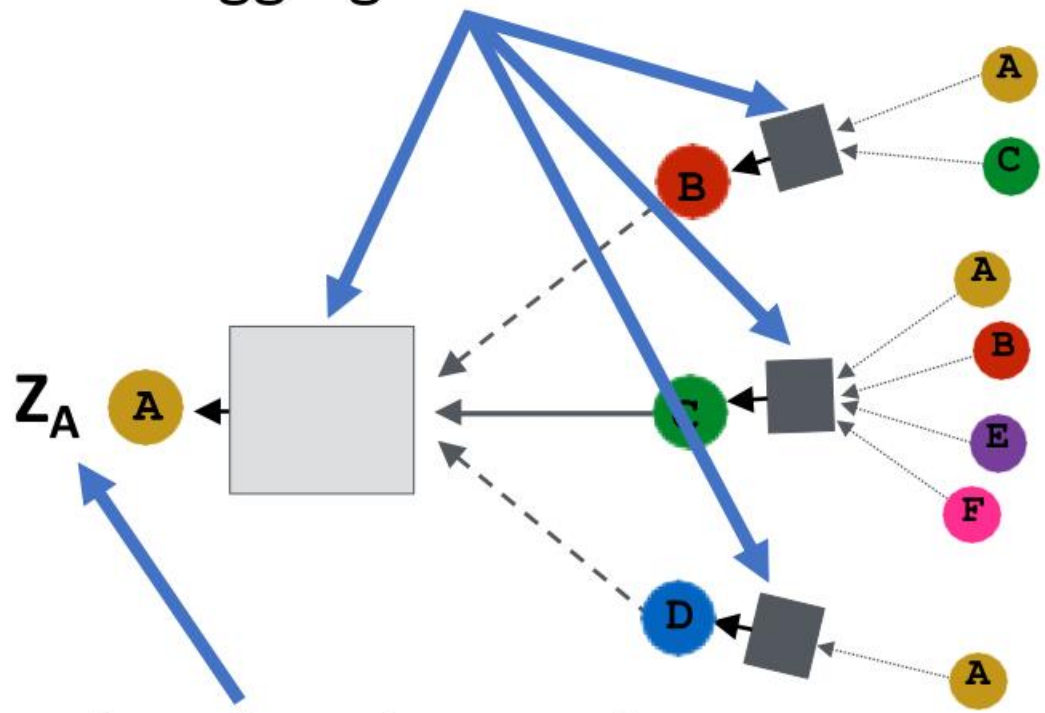


Overview of GNN Model

1) Define a neighborhood aggregation function.

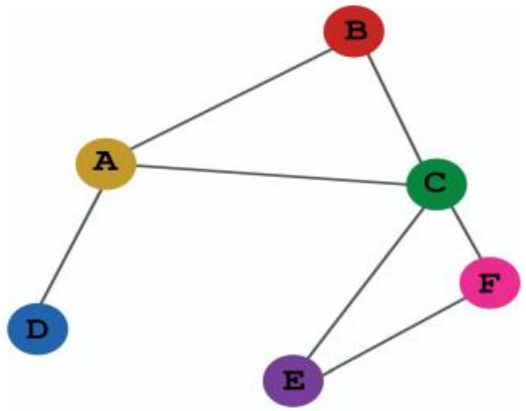


INPUT GRAPH

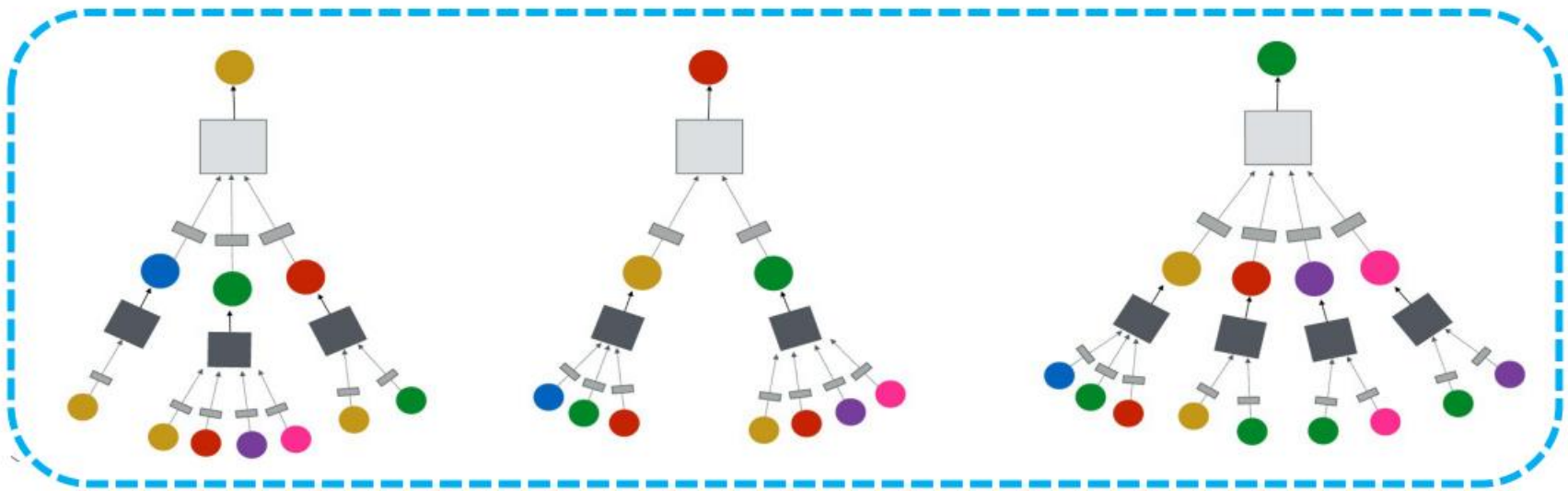


2) Define a loss function on the embeddings, $\mathbf{L(z_v)}$

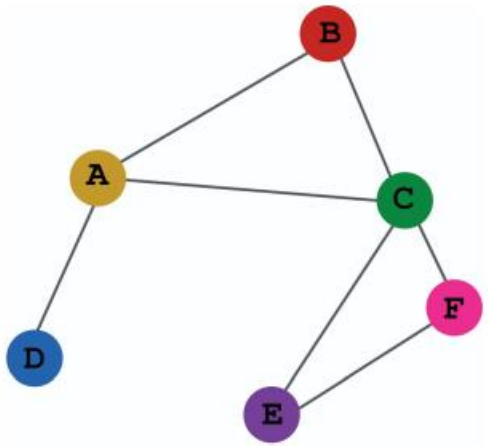
Overview of GNN Model



3) Train on a set of nodes, i.e., a batch of compute graphs



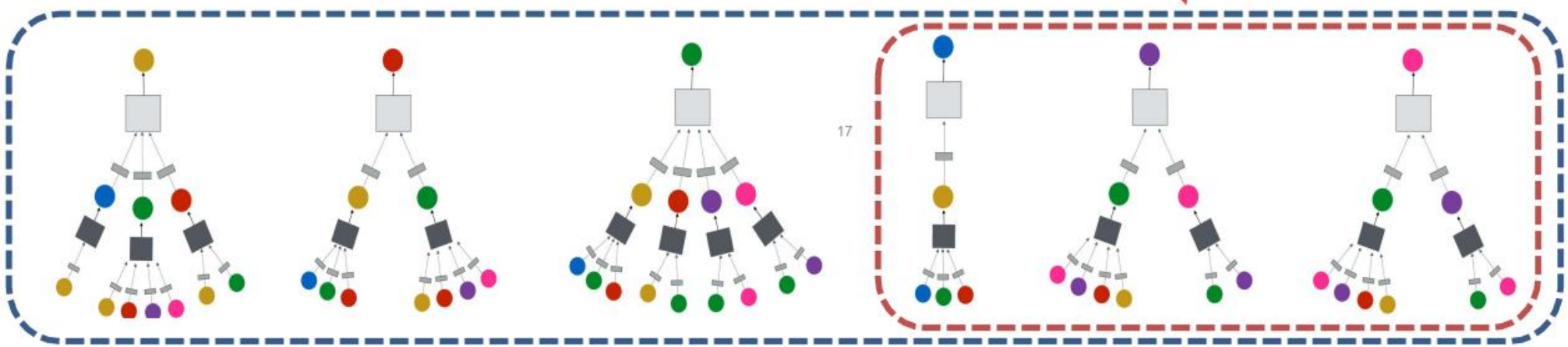
Overview of GNN Model



INPUT GRAPH

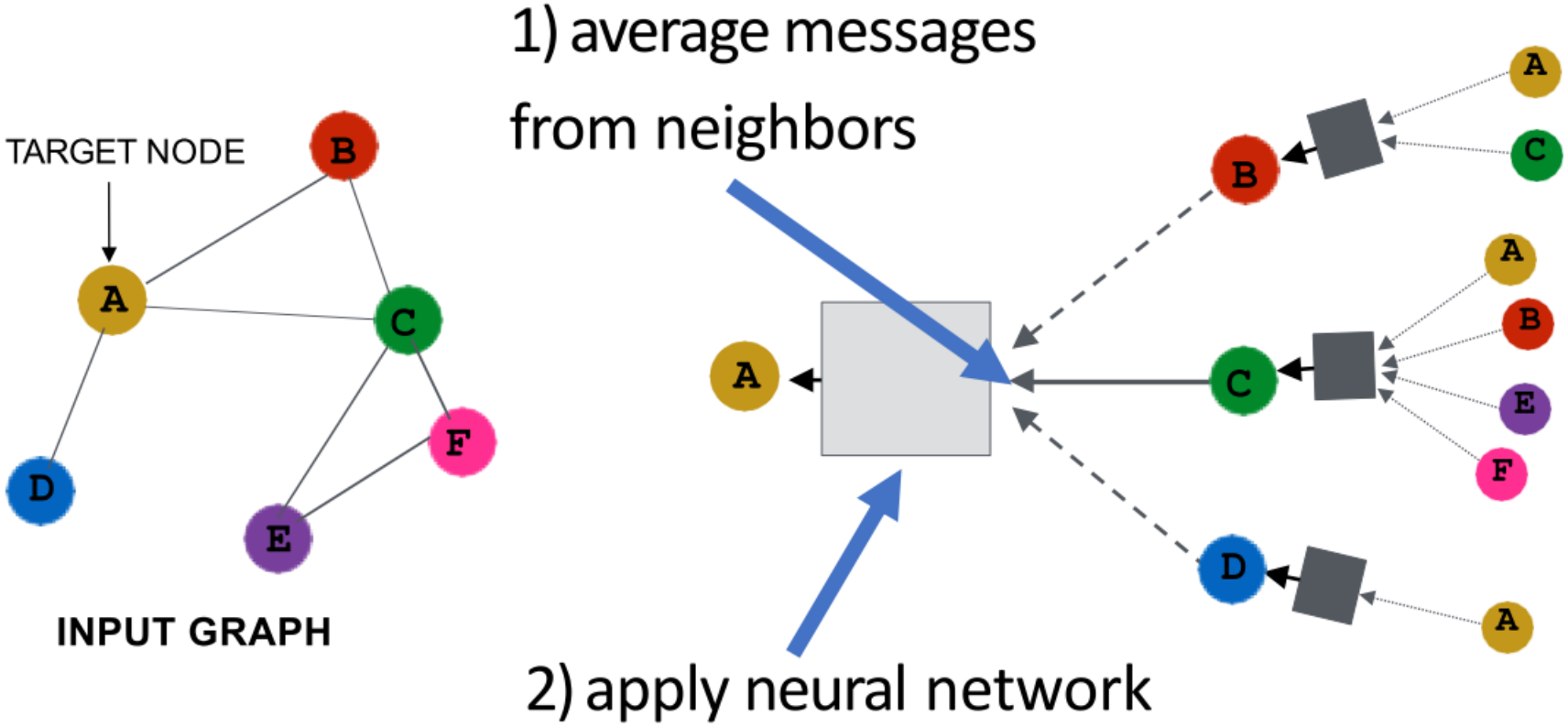
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!



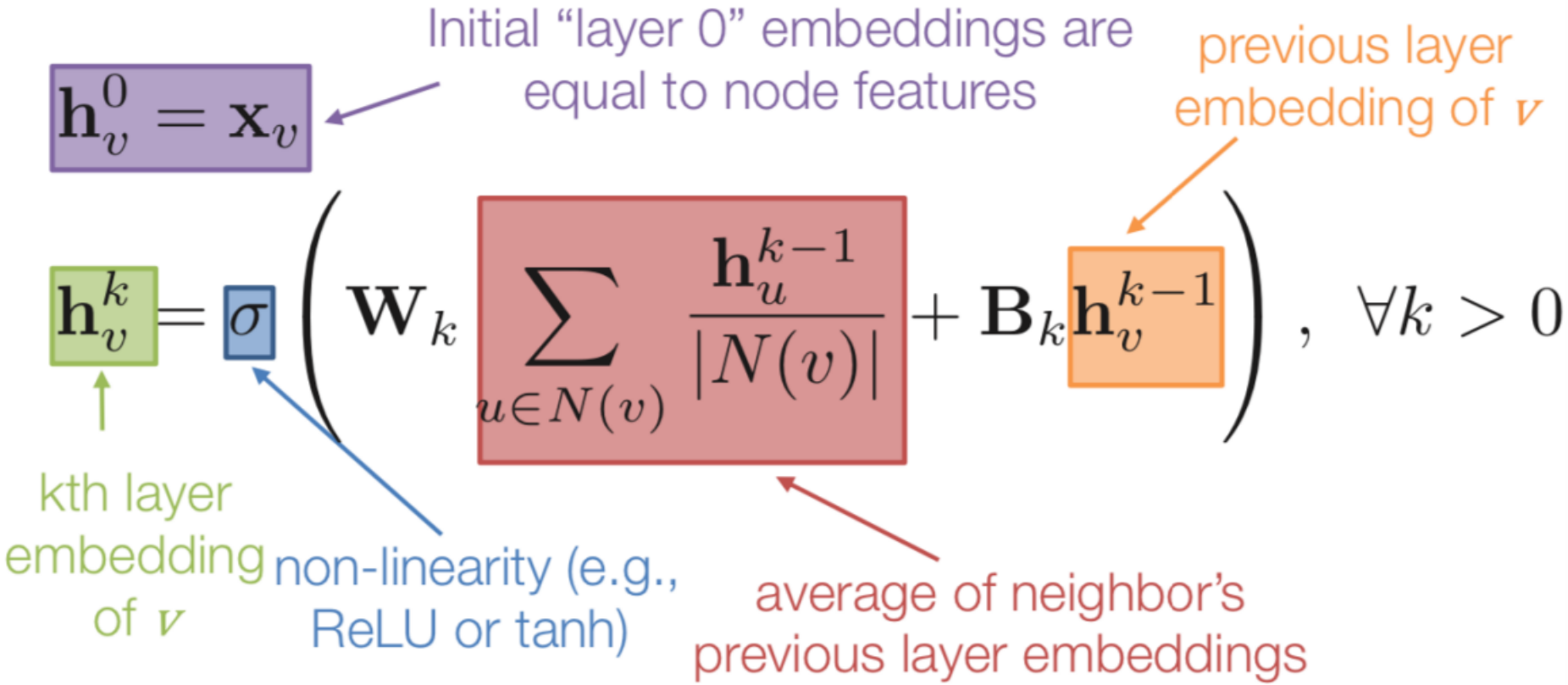
GNN Model: A Case Study

- Basic approach: Average neighbor information and apply a neural network.



GNN Model: A Case Study

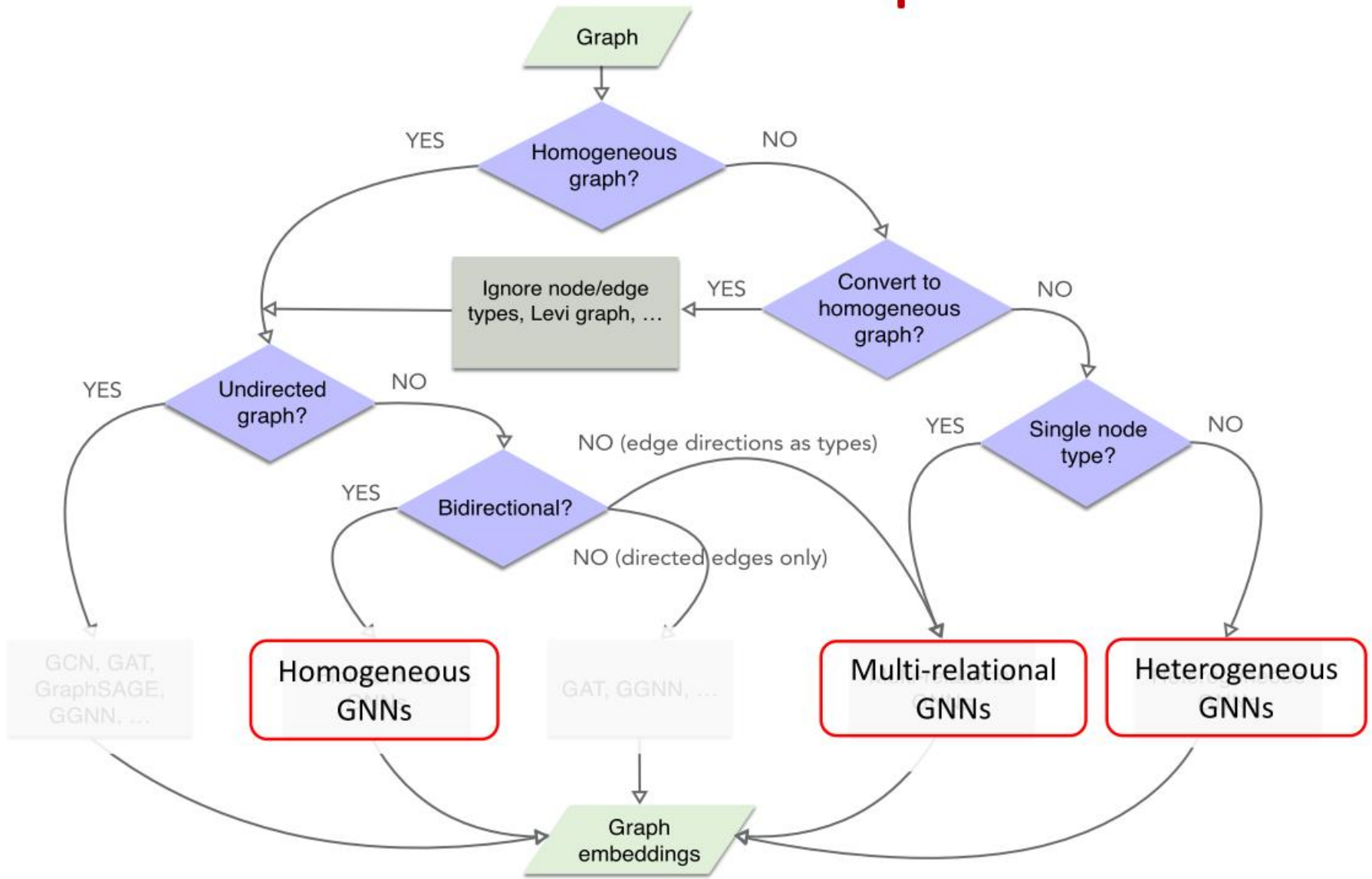
- Basic approach: Average neighbor information and apply a neural network.



GNN Model: Quick Summary

- Key idea: generate node embeddings by aggregating neighborhood information.
  - Allows for parameter sharing in the encoder
  - Allows for inductive learning
- Other state of the art GNNs variants:
  - Graph convolutional networks
  - Graph attention networks
  - Graph isomorphism networks

Which GNNs to Use Given a Graph?



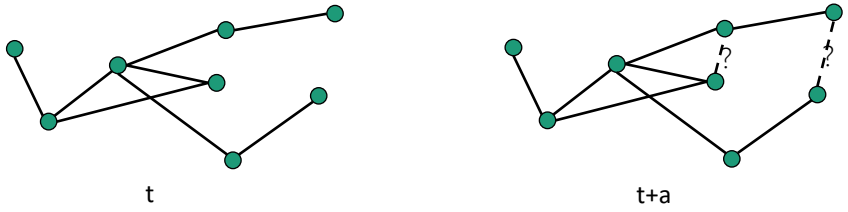
Tasks in graph learning

- Node classification
  - Detect malicious accounts
  - Target right customers
- Link prediction
  - Recommendations
  - Predict missing relations in a knowledge graph
- Graph classification
  - Predict the property of a chemical compound

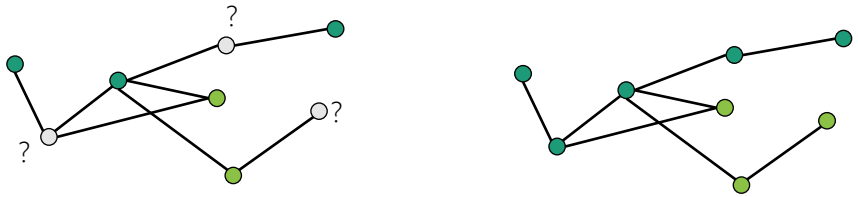
Tasks on Graph-Structured Data

Node-level

Link Prediction

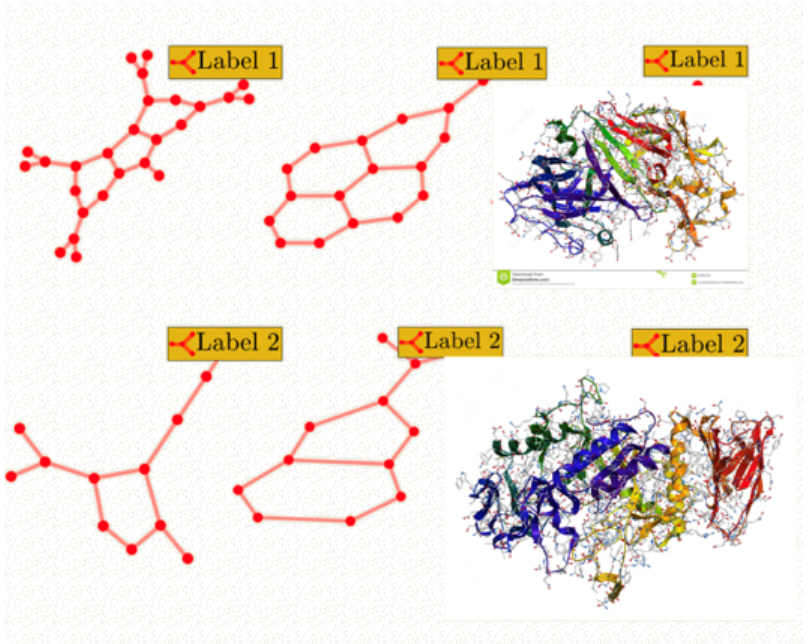


Node Classification



Graph-level

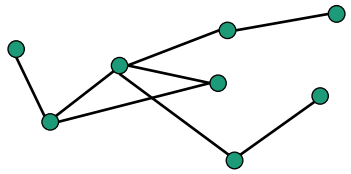
Graph Classification



Tasks on Graph-Structured Data

Node-level

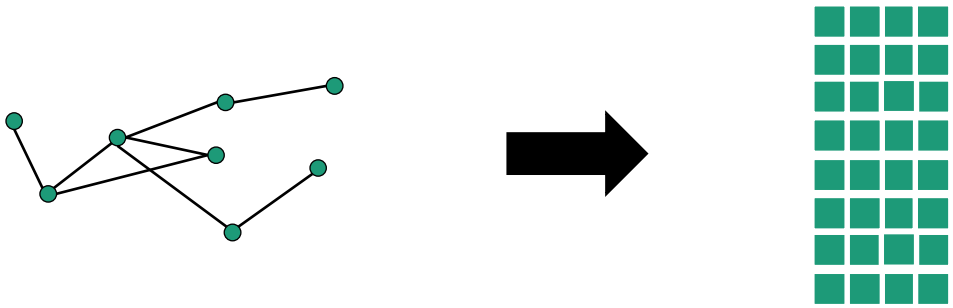
Graph-level



Tasks on Graph-Structured Data

Node-level

Graph-level

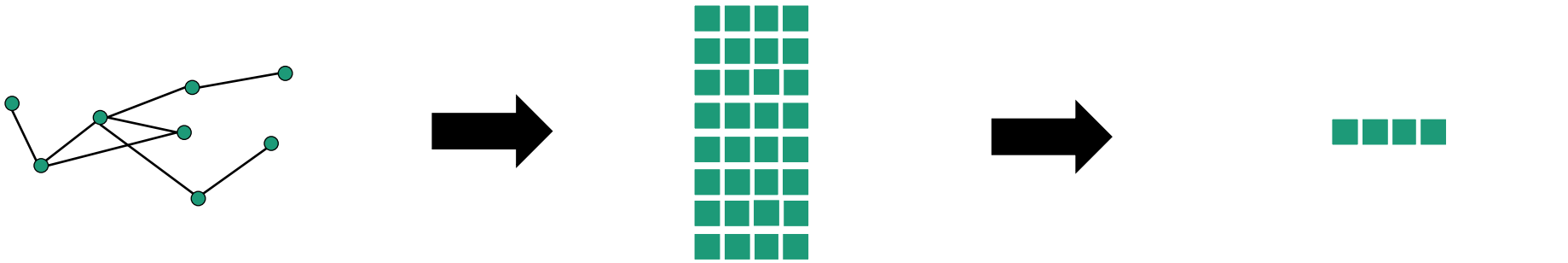


Node Representations

Tasks on Graph-Structured Data

Node-level

Graph-level



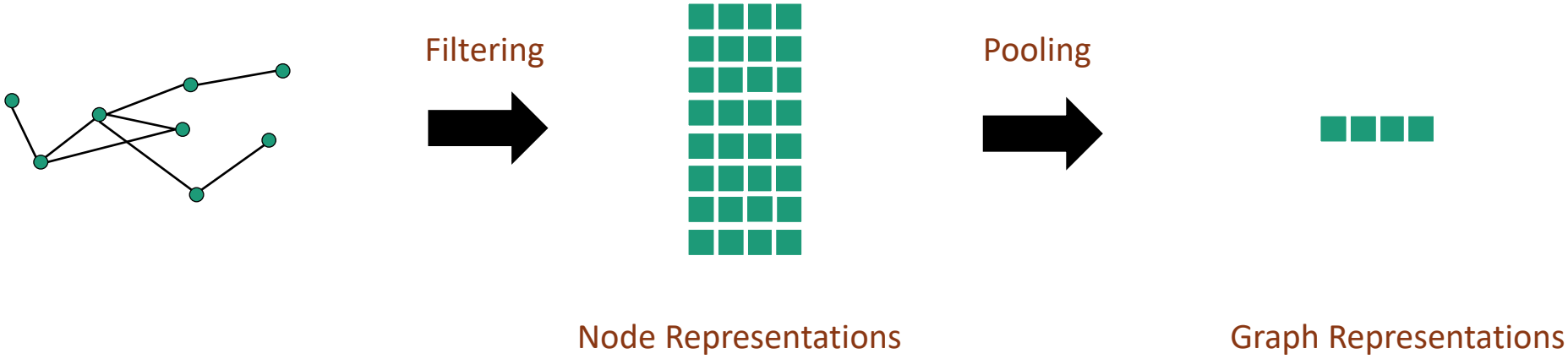
Node Representations

Graph Representation

Tasks on Graph-Structured Data

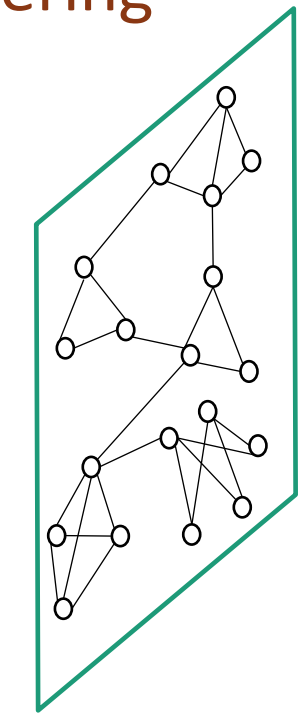
Node-level

Graph-level



Two Main Operations in GNN

Graph Filtering

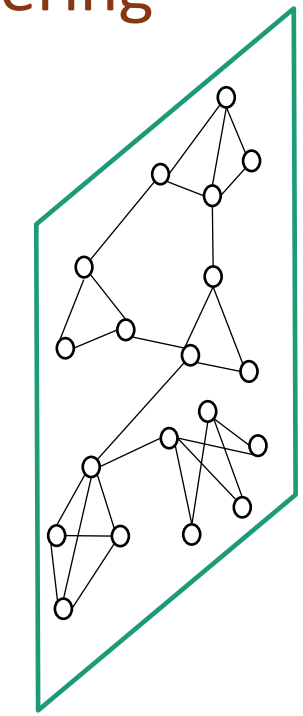


Graph Filtering

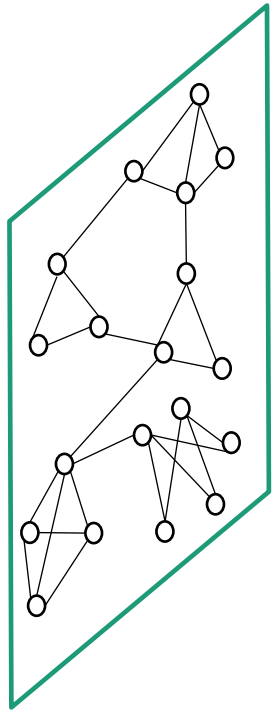
$$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

Two Main Operations in GNN

Graph Filtering



Graph Filtering

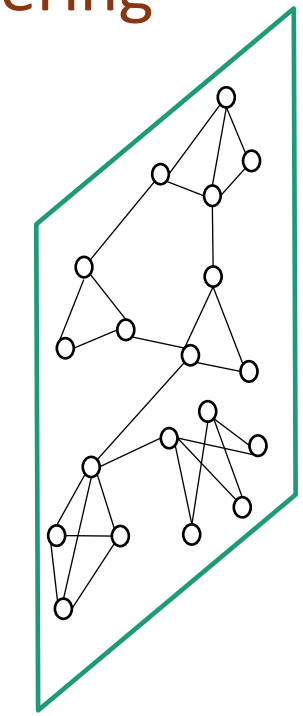


$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$

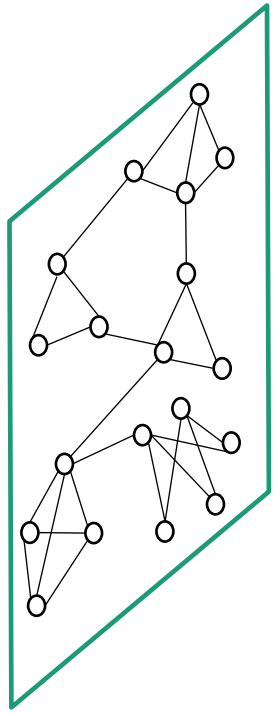
$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X}_f \in \mathbb{R}^{n \times d_{new}}$

Two Main Operations in GNN

Graph Filtering



Graph Filtering



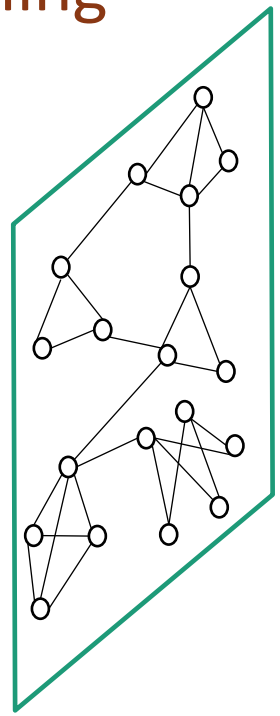
$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$

$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X}_f \in \mathbb{R}^{n \times d_{new}}$

Graph filtering refines the node features

Two Main Operations in GNN

Graph Pooling

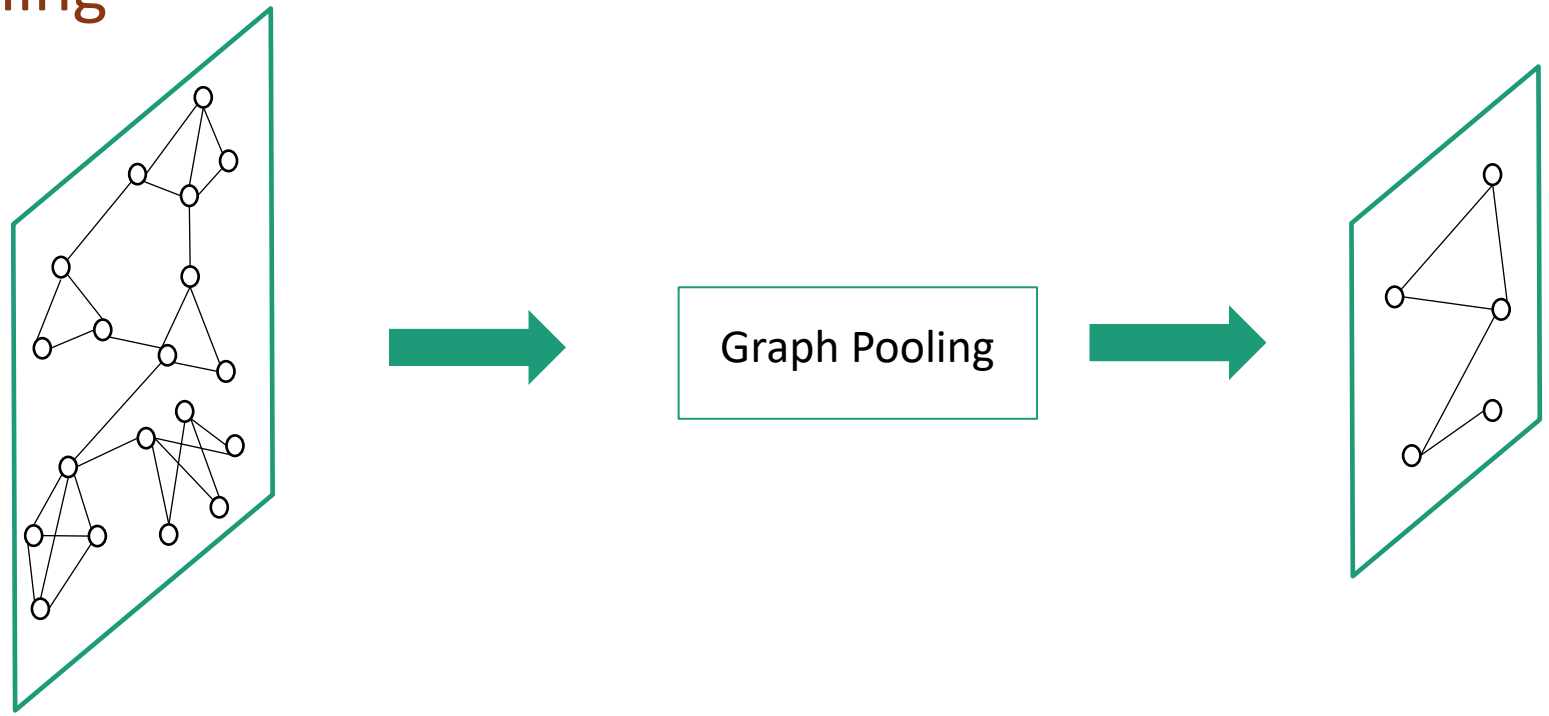


Graph Pooling

$$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

Two Main Operations in GNN

Graph Pooling

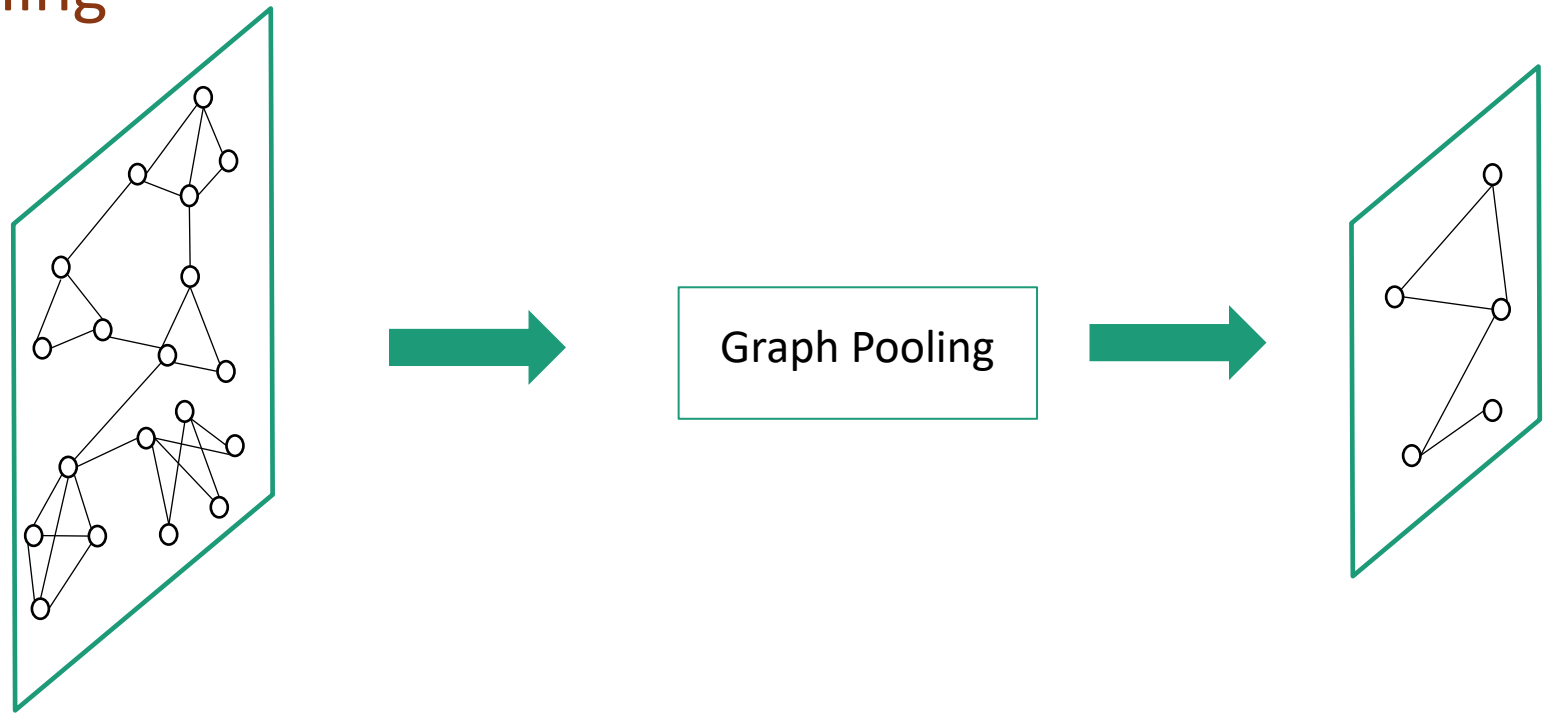


$$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

$$\mathbf{A}_p \in \{0,1\}^{n_p \times n_p}, \mathbf{X}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

Two Main Operations in GNN

Graph Pooling



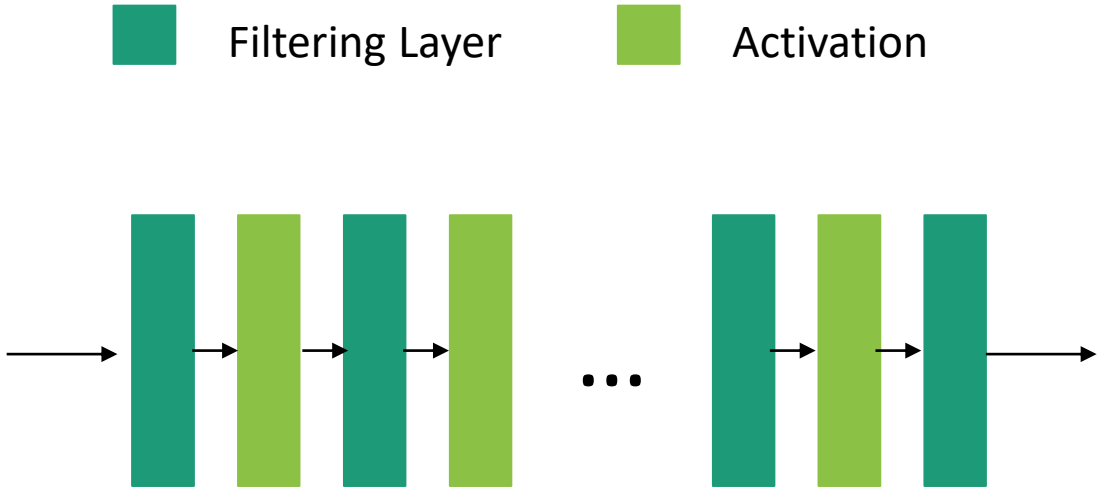
$\mathbf{A} \in \{0,1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$

$\mathbf{A}_p \in \{0,1\}^{n_p \times n_p}, \mathbf{X}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$

Graph pooling generates a smaller graph

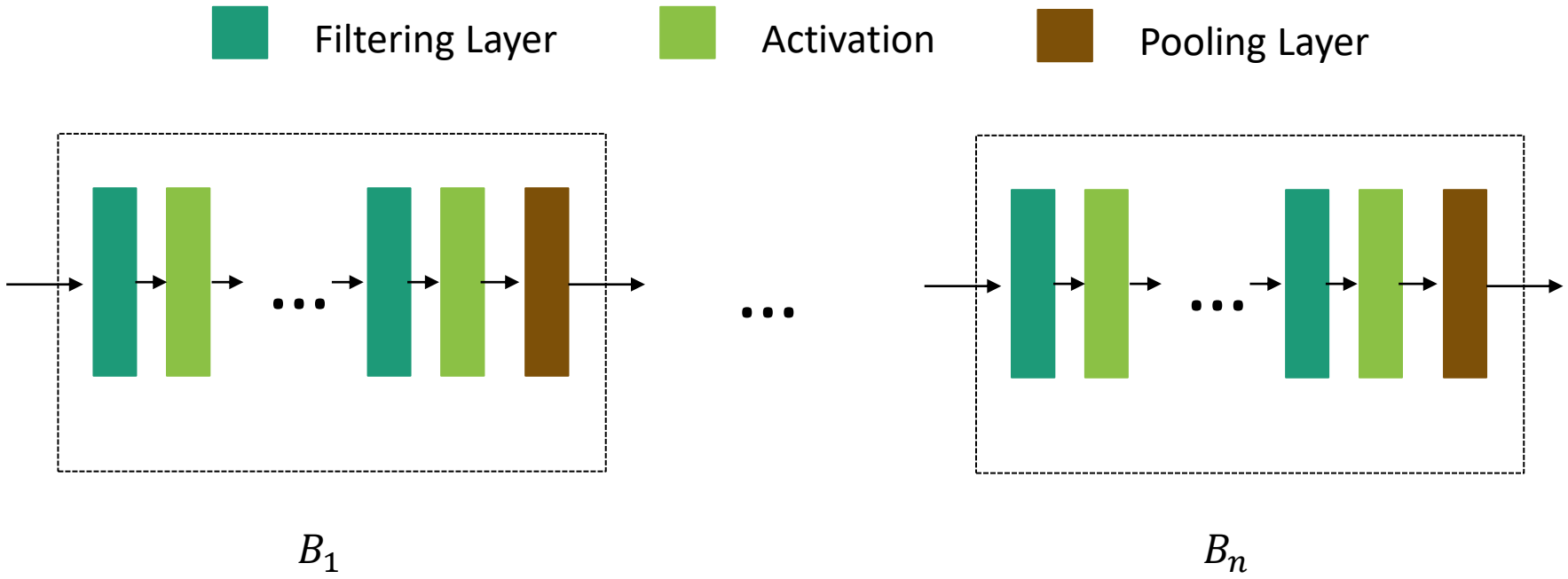
General GNN Framework

For node-level tasks



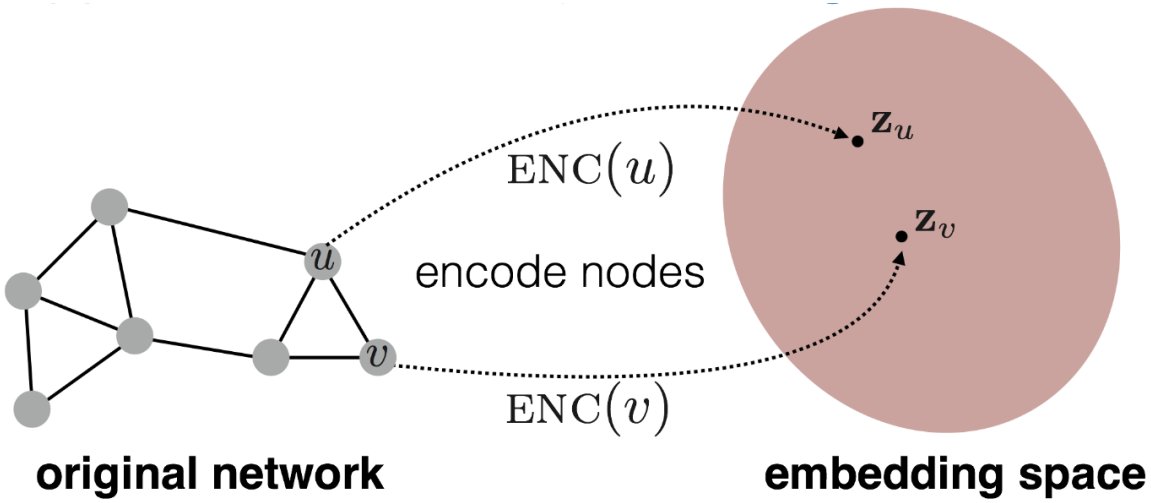
General GNN Framework

For graph-level tasks



Graph learning and node embeddings

- Embed nodes to a low-dimension space so that these embeddings capture the essential task-specific information and use them to train off-the-self classifiers.
  - For example, node similarities in the embedding space approximates the similarities in the original graph.



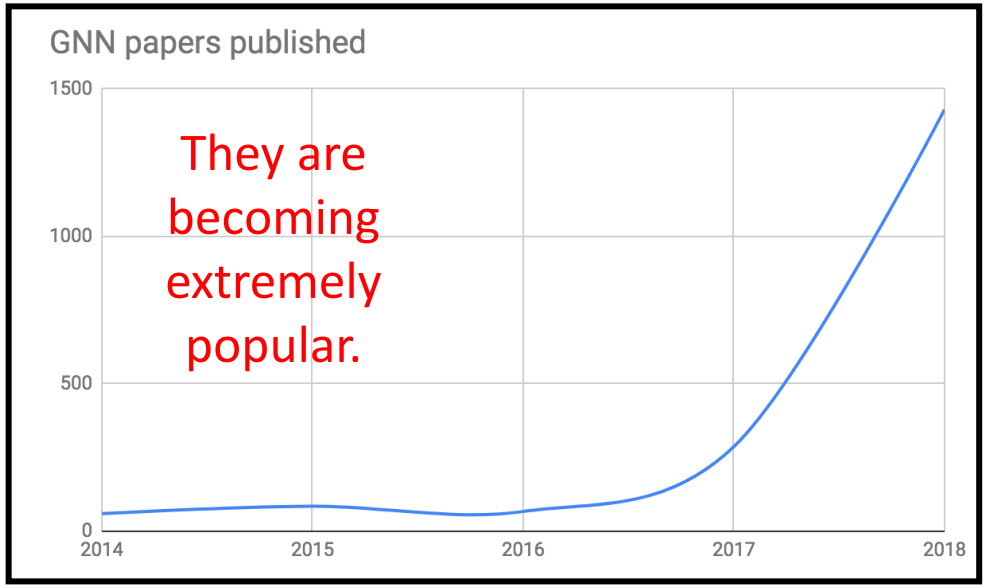
Traditional graph learning approaches

- Generate embeddings by manual feature engineering
  - Requires domain expertise, involves considerable manual fine-tuning, time consuming, does not scale, …
- Automatically generate embeddings using unsupervised dimensionality reduction approaches
  - Singular value decomposition, tensor decomposition, co-factorization, deep walks, etc.
  - Cannot effectively combine rich attributes with network structure.
  - Employ mostly (multi-)liner models.
  - Do not allow for end-to-end learning.

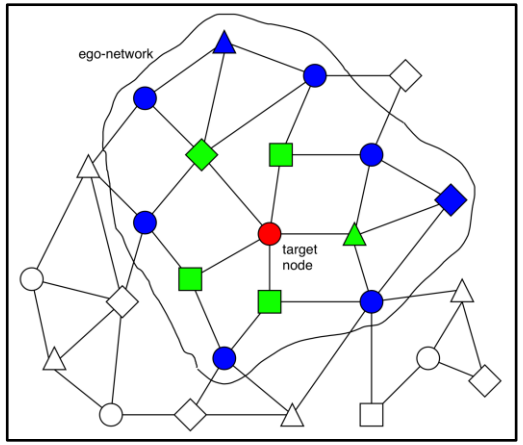
Can we do better?

Graph Neural Network (GNN)

A family of (deep) neural networks that learn node, edge, and graph embeddings.



How do GNNs work?



An ego-network around each node is used to learn an embedding that captures task-specific information.

The embeddings use both the structure of the graph and the features of the nodes and edges.

The embeddings are learned in an end-to-end fashion; thus, the predictions are a function of the target node's ego-network.

A general graph neural network formalism

Graph neural networks are based on message-passing

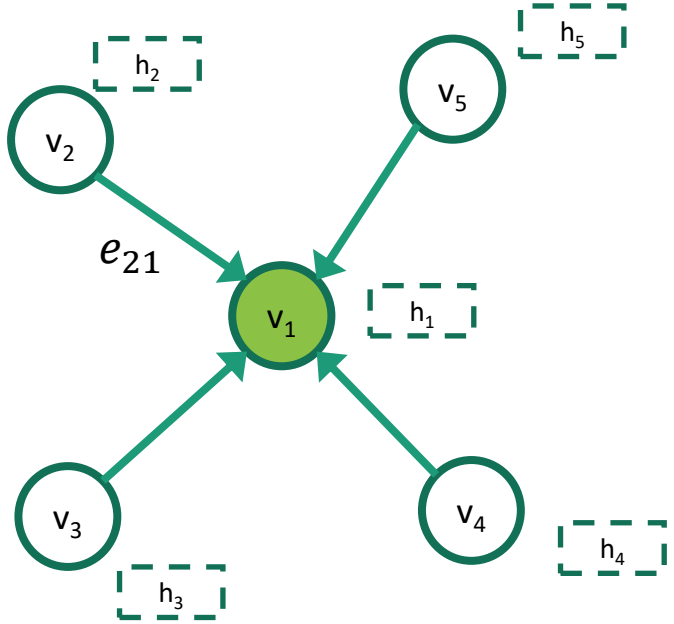
Reduce/Aggregate

Message

$$m_v^{(l)} = \sum_{w \in N(v)} M^{(l)}(h_v^{(l-1)}, h_w^{(l-1)}, e_{vw})$$

$$h_v^{(l)} = U^{(l)}(h_v^{(l-1)}, m_v^{(l)})$$

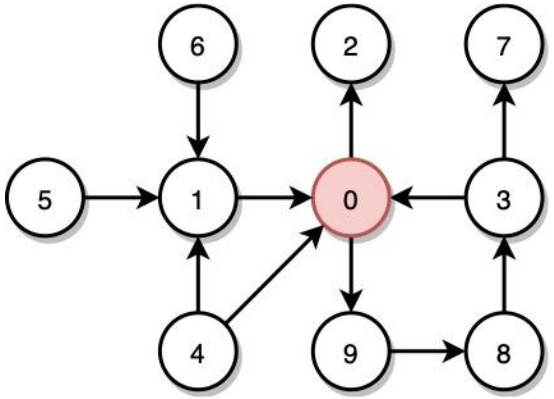
Update

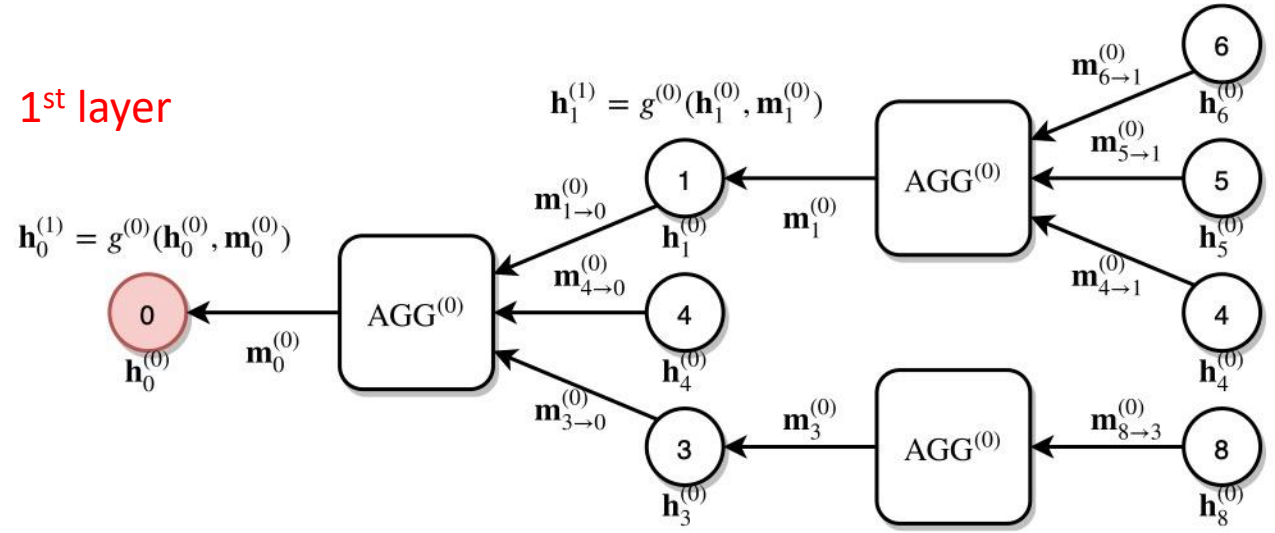


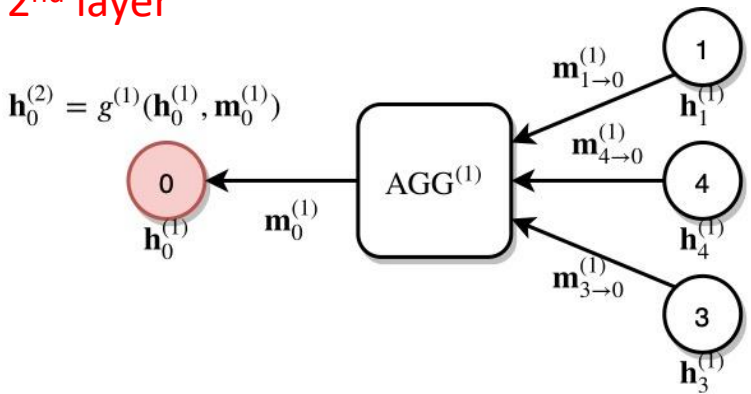
Neural Message Passing for Quantum Chemistry

A multi-layer GNN

Multiple GNN layers can be stacked together.

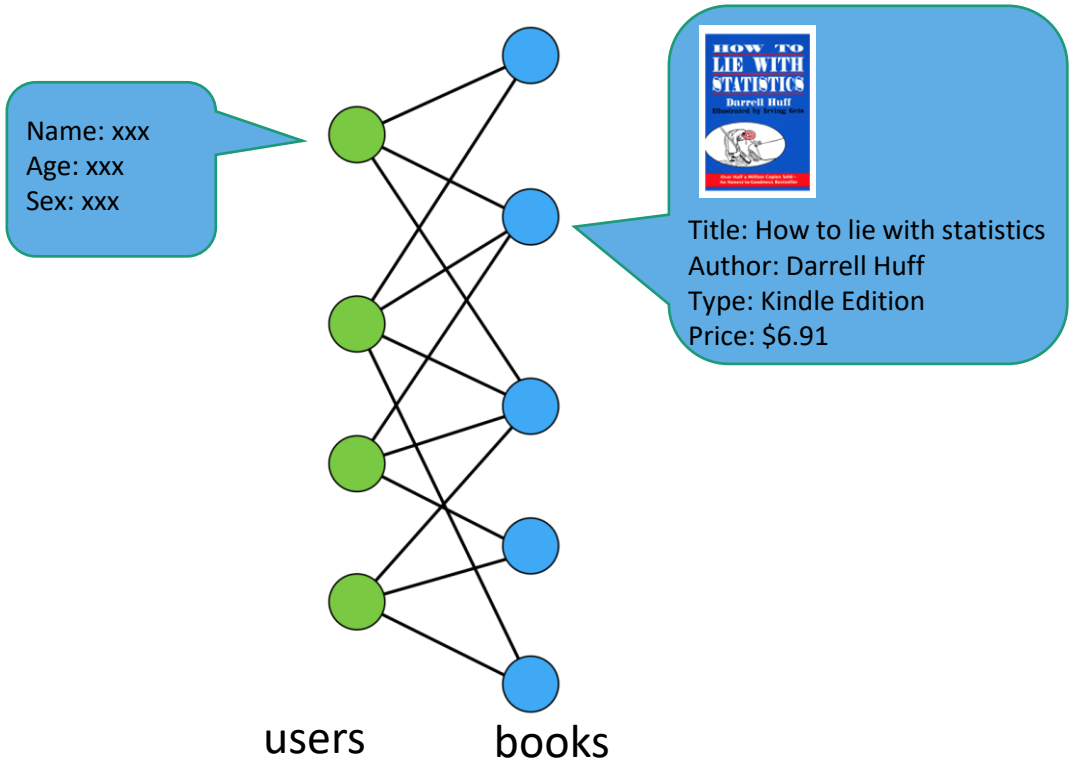






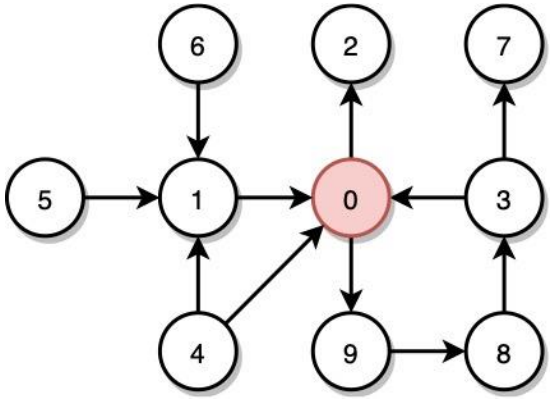
Why are graph neural networks better?

GNNs compute node embeddings using both the structure of the graph and the features of the nodes and edges.

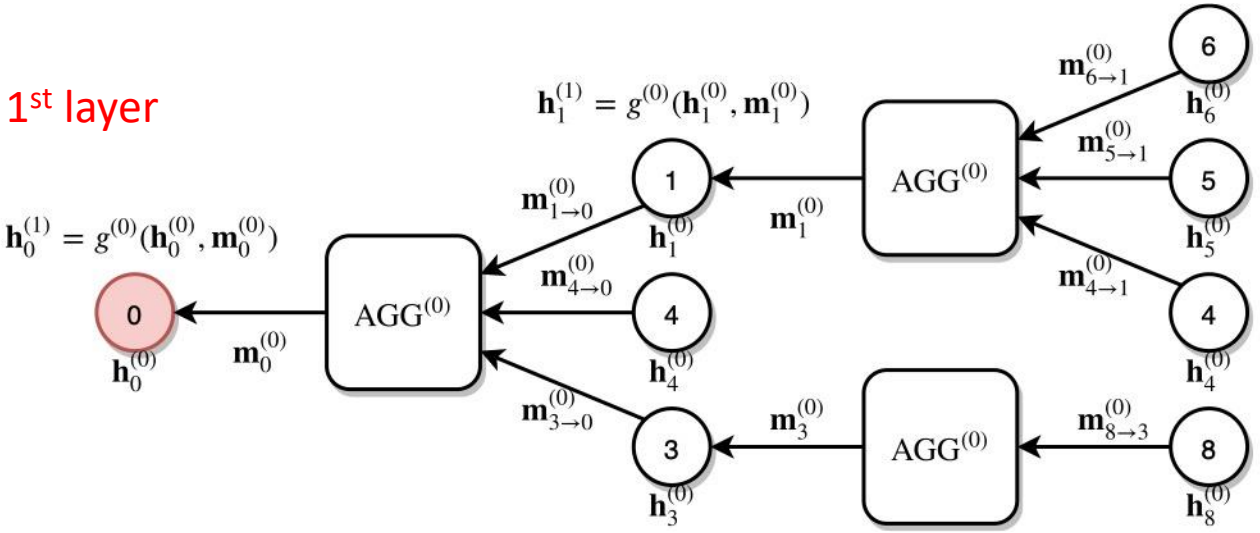


Why are graph neural networks better?

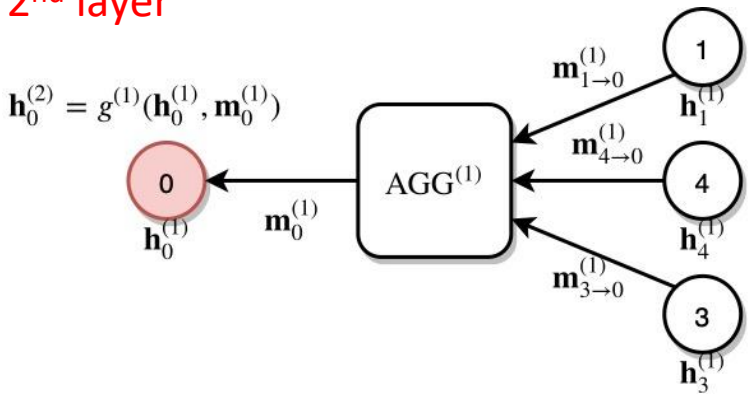
GNNs can *integrate* topologically distant information in a non-linear fashion.



1st layer

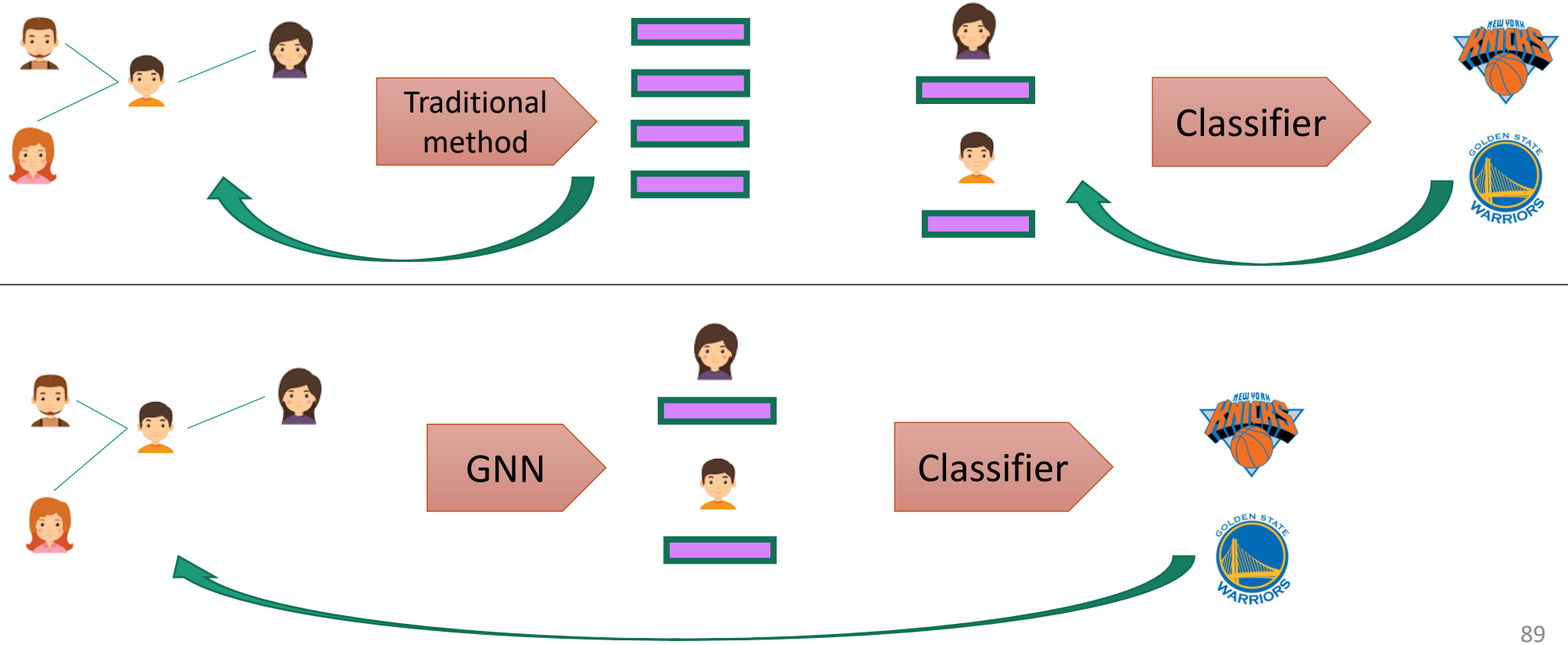


2nd layer



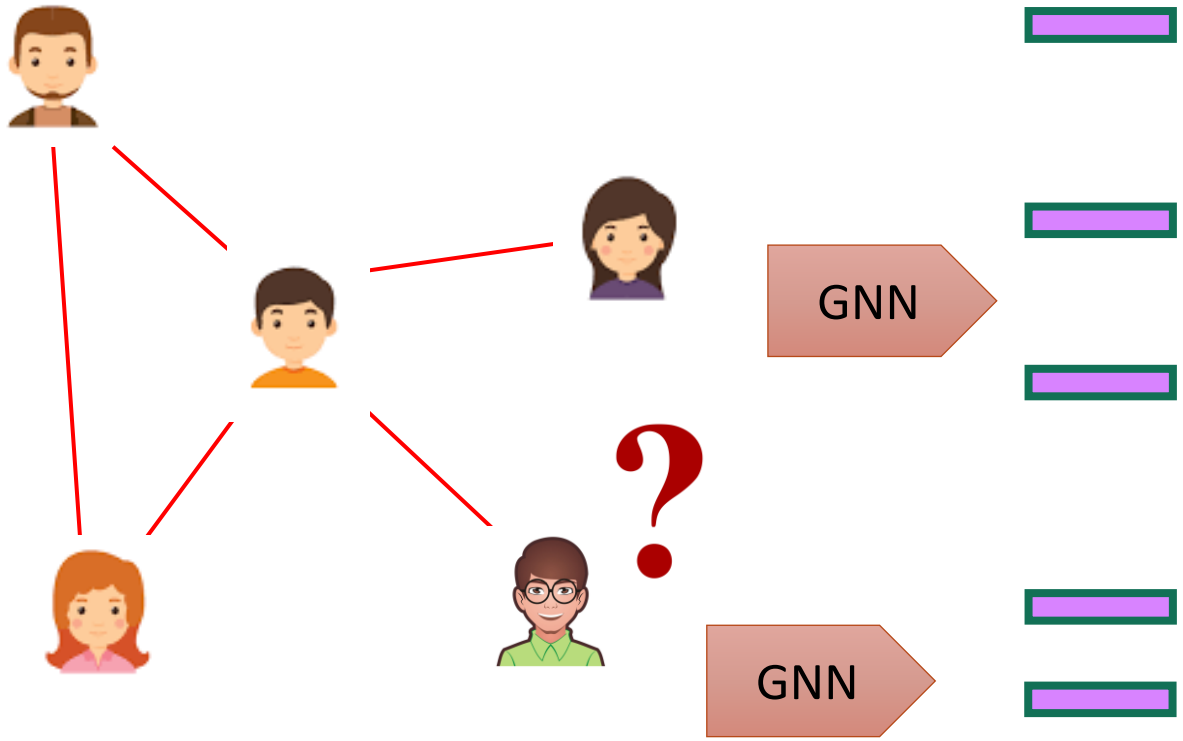
Why are graph neural networks better?

GNNs and the downstream classification/regression models can be trained in an end-to-end fashion.



Why are graph neural networks better?

GNNs are naturally inductive because they learn the same neural networks on all the nodes and edges.

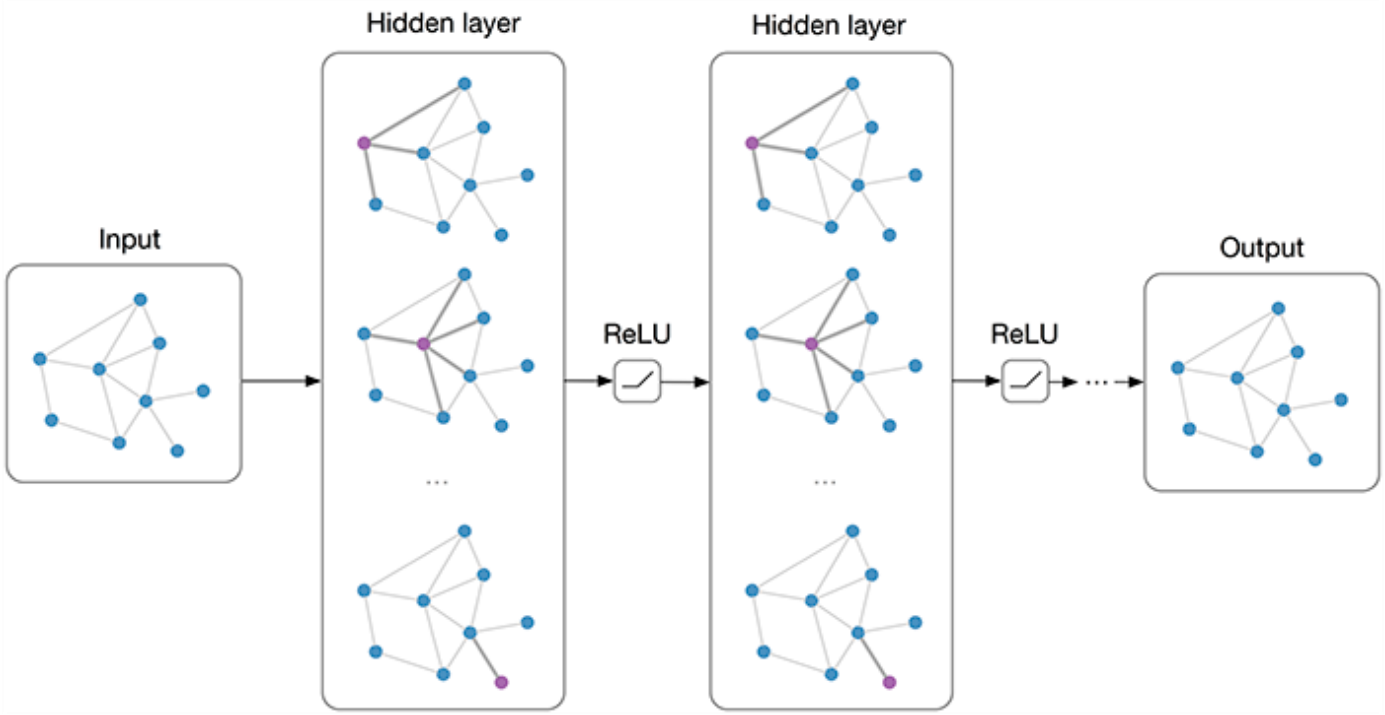


Example 1: Graph convolution network (GCN)

$$M_{vw}^{(l)} = \frac{h_w^{(l-1)}}{d_v+1}$$

$$m_v^{(l)} = \sum_{w \in N(v) \cup \{v\}} M_{vw}^{(l)}$$

$$h_v^{(l)} = \phi(m_v^{(l)} W^{(l)})$$



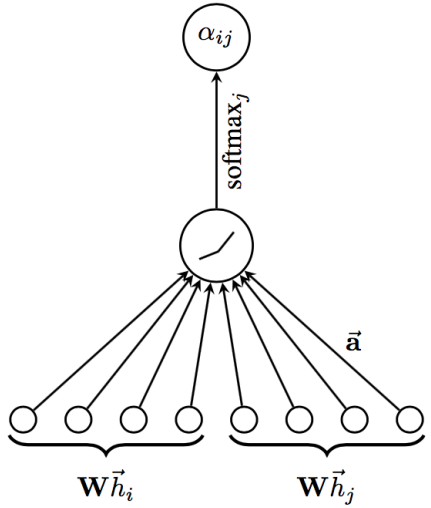
Example 2: Graph attention networks (GAT)

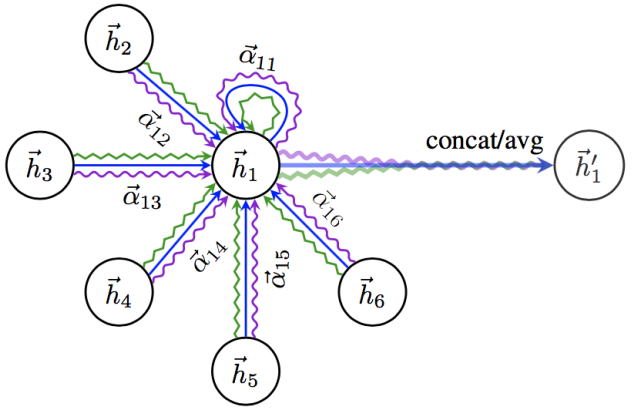
GAT provides weighted sum over the neighborhood—Enables to selectively integrate information.

$$M_{vw}^{(l)} = \alpha_{vw} h_w^{(l-1)}$$

$$m_v^{(l)} = \sum_{w \in N(v) \cup \{v\}} M_{vw}^{(l)}$$

$$h_v^{(l)} = \phi(m_v^{(l)} W^{(l)})$$

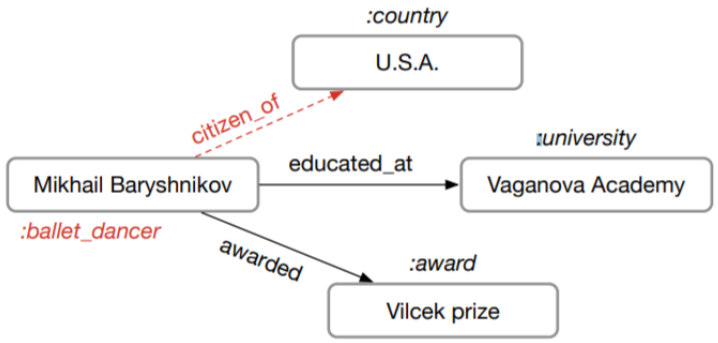




$$\alpha_{vw} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T [W\vec{h}_v || W\vec{h}_w]))}{\sum_{k \in N_v} \exp(\text{LeakyReLU}(\vec{a}^T [W\vec{h}_v || W\vec{h}_k]))}$$

Example 3: Relational graph convolution networks (RGCN)

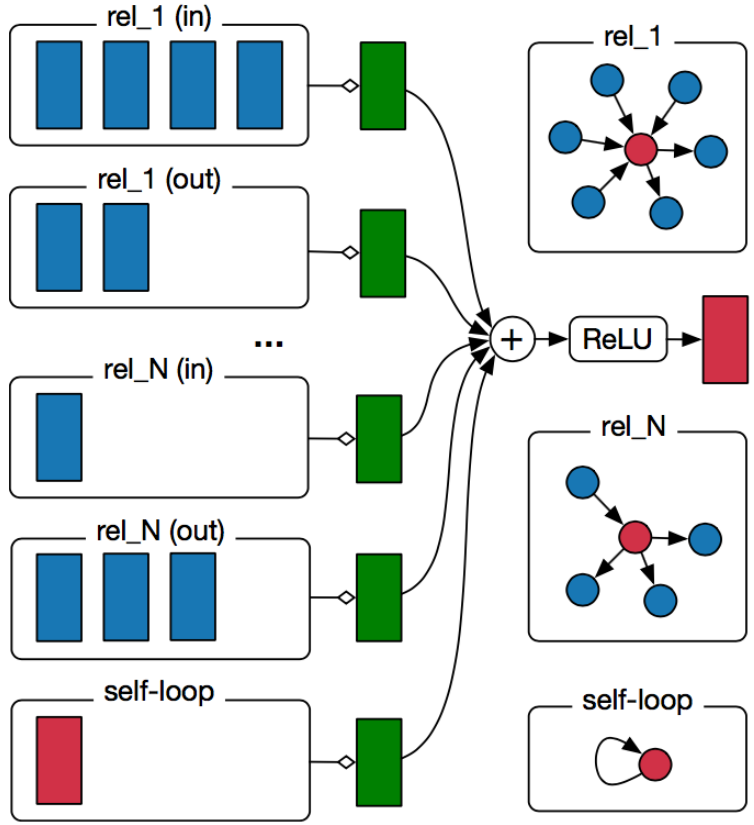
Handles graphs whose nodes are connected with different relations.



$M_{vw}^{(l)} = \frac{1}{c_{v,r}} W_r^{(l)} h_w^{(l-1)}$, r is the relation of e_{vw}.

$$m_v^{(l)} = \sum_{w \in N(v) \cup \{v\}} M_{vw}^{(l)}$$

$$h_v^{(l)} = \sigma(m_v^{(l)} W^{(l)})$$



How to train an end-to-end GNN?

- An L-layer GNN computes node embeddings

$$m_v^{(l)} = \sum_{w \in N(v)} M^{(l)}(h_v^{(l-1)}, h_w^{(l-1)}, e_{vw})$$

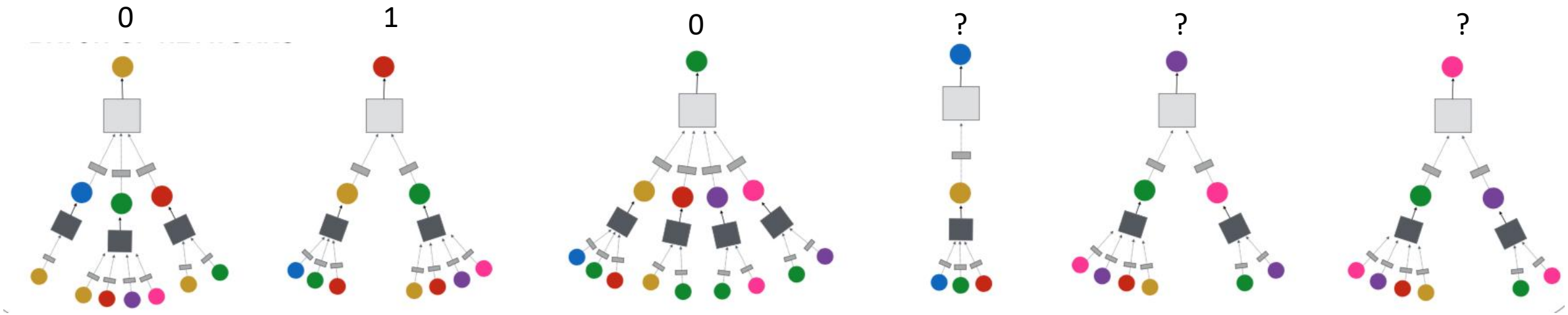
$$h_v^{(l)} = U^{(l)}(h_v^{(l)}, m_v^{(l)})$$

- We use $h_v^{(l)}$ in a downstream model trained with any loss function.

Node classification with GNN

- Node classification is trained in the semi-supervised setting.

$$\text{loss} = \text{CrossEntryLoss}(h_v^{(l)} W, label_v)$$



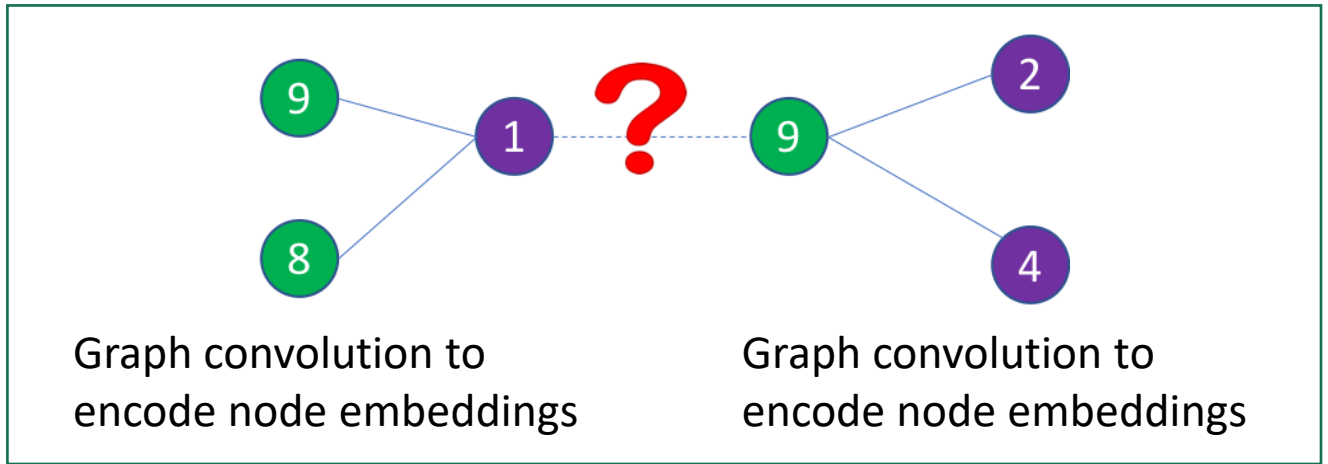
Link prediction with GNN

- We train a link prediction model with connectivity of nodes as the training signal.
  - Positive edges are trained against a few negative edges

$$\text{loss} = -\log\left(\sigma(h_v^T h_w)\right) - Q \cdot E_{u_n \sim P_n(v)} \log\left(\sigma(-h_v^T h_{u_n})\right)$$

Positive edges

Negative edges

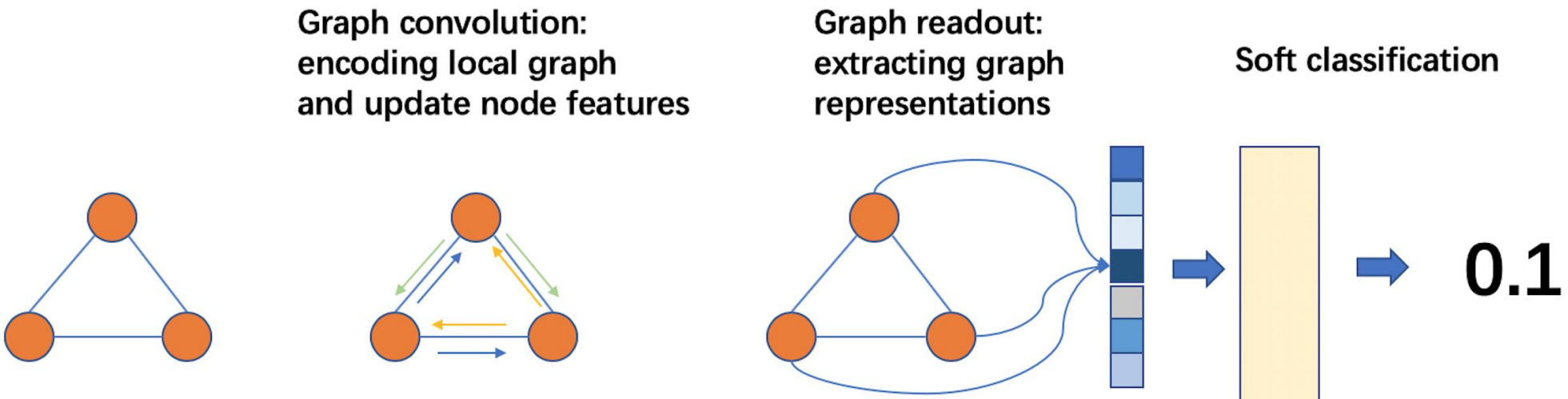


Graph classification

- Graph readout to compute graph embeddings.
- Train a graph classifier on the graph embedding.

$$g = readout\left(h_1^{(l)}, h_2^{(l)}, \ldots, h_n^{(l)}\right)$$

$$\text{loss} = \text{CrossEntryLoss}(g\,W, label)$$

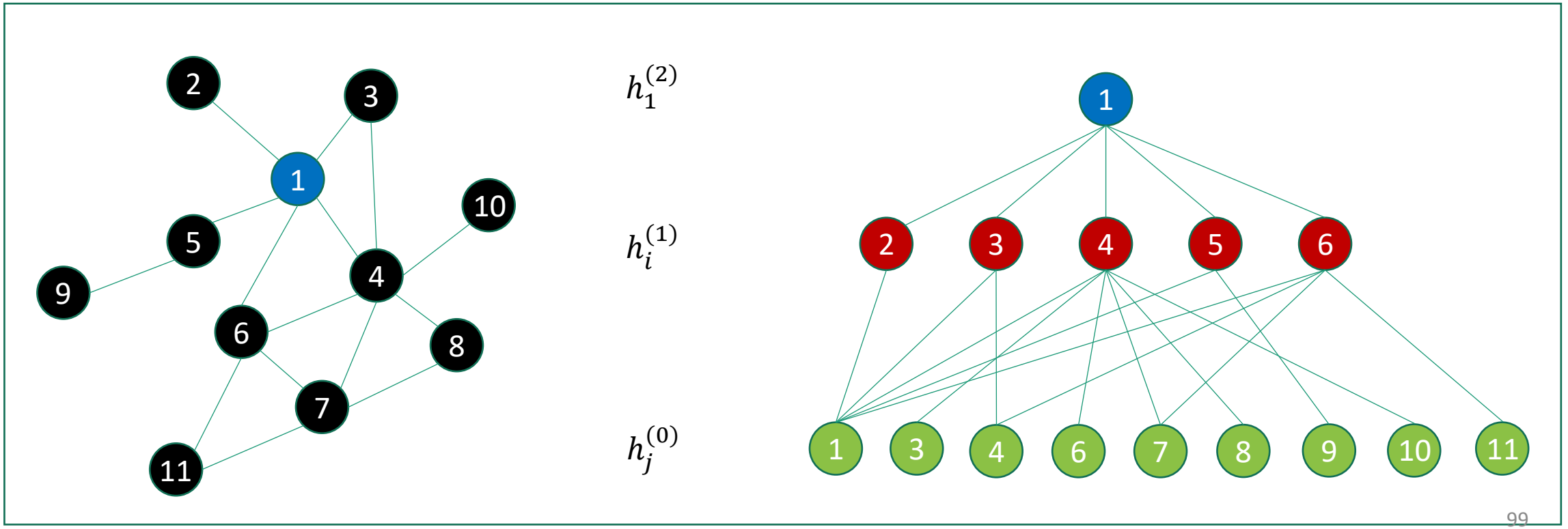


How to train GNN on large graphs

- Many applications have extremely large graphs—millions of nodes and billions of edges:
  - Social networks
  - Recommendation
  - Knowledge graph
  - …
- A typical training method: mini-batch training

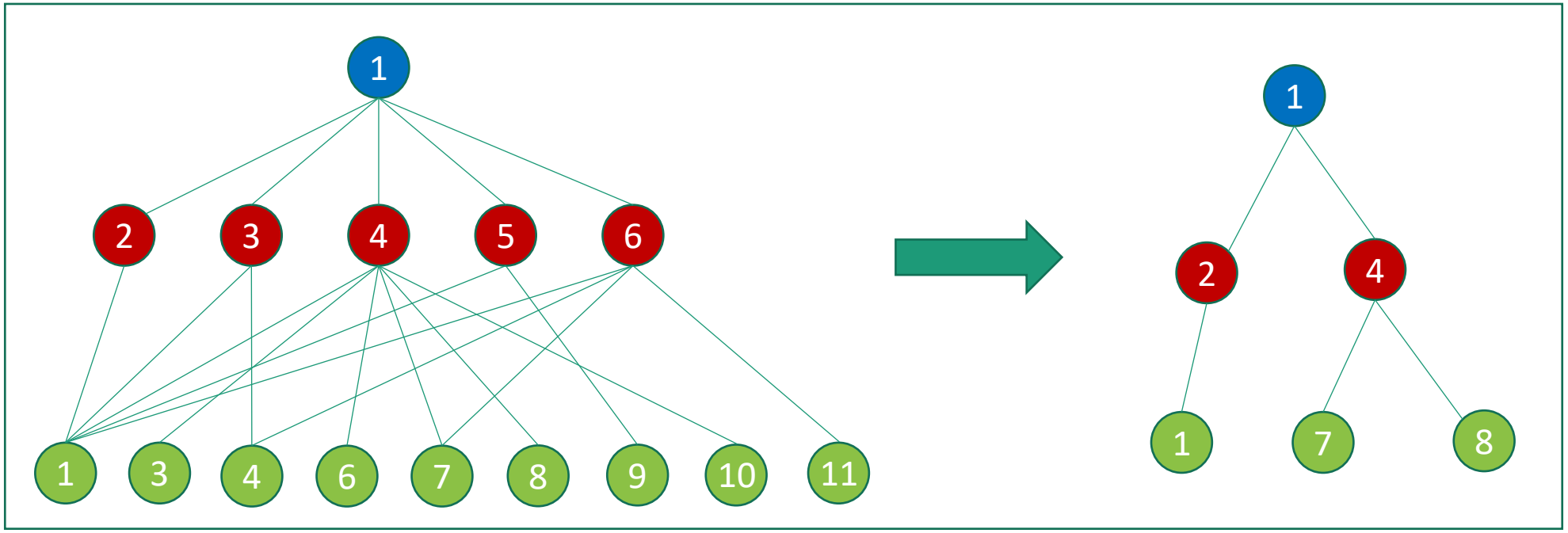
A glance of mini-batch training on graphs

- A mini batch represents the computation graph for target nodes.
- Small-world graphs lead to a huge computation graph.



Mini-batch with neighbor sampling

- Prune the computation graph:
  - Sample neighbors from a neighbor list of a vertex.



Graph Neural Networks: Popular Models

- Spectral-based Graph Filters
  - GCN (Kipf & Welling, ICLR 2017), Chebyshev-GNN (Defferrard et al. NIPS 2016)
- Spatial-based Graph Filters
  - MPNN (Gilmer et al. ICML 2017), GraphSage (Hamilton et al. NIPS 2017)
  - GIN (Xu et al. ICLR 2019)
- Attention-based Graph Filters
  - GAT (Velickovic et al. ICLR 2018)
- Recurrent-based Graph Filters
  - GGNN (Li et al. ICLR 2016)

Graph Convolution Networks (GCN)

Key idea: spectral convolution on graphs

Eigen-decomposition is expensive

$$f_{\text{filter}} * \mathbf{x}_i = \mathbf{U} f(\mathbf{\Lambda}) \mathbf{U}^T \mathbf{x}_i$$

Chebyshev polynomials accelerates but still not powerful

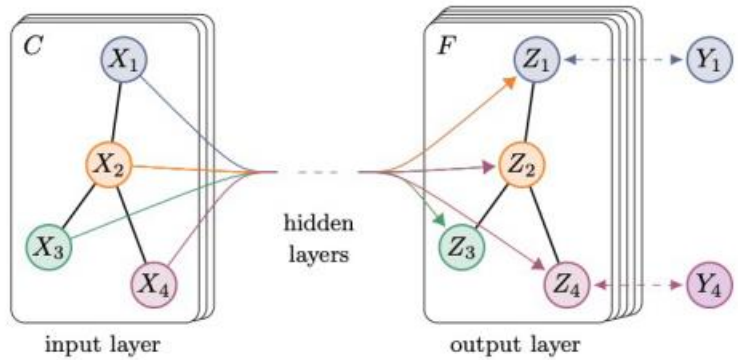
$$f'_{\text{filter}} * \mathbf{x}_i \approx \sum_{p=0}^{P} \theta'_p \mathbf{T}_p(\tilde{\mathbf{L}}) \mathbf{x}_i$$

First-order approximation fast and powerful

$$f_{\text{filter}} * \mathbf{h}_i^{(l)} \approx \theta(I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) \mathbf{h}_i^{(l)}$$

Renormalization trick stabilizes the numerical computation

$$\mathbf{H}^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)})$$



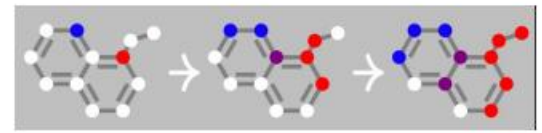
GCN in NLP Tasks:

- Text classification
- Question Answering
- Text Matching
- Topic Modeling
- Information Extration

Message Passing Neural Network (MPNN)

Key idea: graph convolutions as a message passing process

MPNN: $$\mathbf{h}_i^{(l)} = f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = f_U(\mathbf{h}_i^{(l-1)}, \sum_{v_j \in N(v_i)} f_M(\mathbf{h}_i^{(l-1)}, \mathbf{h}_j^{(l-1)}, \mathbf{e}_{i,j}))$$



expensive if the number of nodes are large

Update and aggregation functions

Node and edge embeddings

GraphSage: $$f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = \sigma(\mathbf{W}^{(l)} \cdot f_M(\mathbf{h}_i^{(l-1)}, \{\mathbf{h}_j^{(l-1)}, \forall v_j \in N(v_i)\}))$$

sampling to obtain a fixed number of neighbors

Aggregation functions

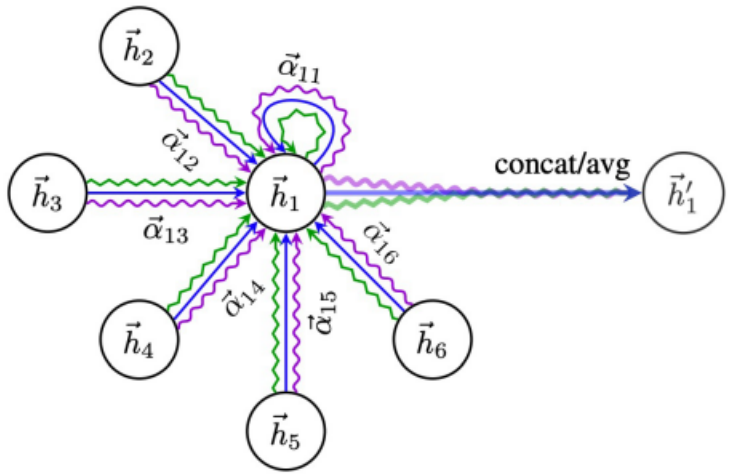
Node embeddings

MPNN and GraphSage in NLP Tasks:

- Knowledge graph
- Information extraction
- Semantic parsing

Graph Attention Network (GAT)

Key idea: dynamically learn the weights (attention scores) on the edges when performing message passing



Weighted sum of node embeddings

$$\mathbf{h}_i^{(l)} = f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = \sigma\Big(\sum_{v_j \in N(v_i)} \alpha_{ij} \mathbf{W}^{(l)} \mathbf{h}_j^{(l-1)}\Big)$$

Learned local weights with self-attention

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{u}^{(l)^T}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} || \mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)}]))}{\sum_{v_k \in N(v_i)} \exp(\text{LeakyReLU}(\mathbf{u}^{(l)^T}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} || \mathbf{W}^{(l)}\mathbf{h}_k^{(l-1)}]))}$$

Intermediate node embeddings

$$f_{\text{filter}}(A, \mathbf{H}^{(l-1)}) = ||_{k=1}^{K} \sigma\Big(\sum_{v_j \in N(v_i)} \alpha_{ij}^k \mathbf{W}_k^{(l)} \mathbf{h}_j^{(l-1)}\Big)$$

Final node embeddings

$$f_{\text{filter}}(A, \mathbf{H}^{(L-1)}) = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\sum_{v_j \in N(v_i)} \alpha_{ij}^k \mathbf{W}_k^{(L)} \mathbf{h}_j^{(L-1)}\Big)$$

GAT in NLP Tasks:

- Text classification
- Question Answering
- Knowledge graph
- Information extraction
- Semantic parsing

Gated Graph Neural Networks (GGNN)

Key idea: the use of Gated Recurrent Units while taking into account edge type and directions

Zero-padding input node embeddings

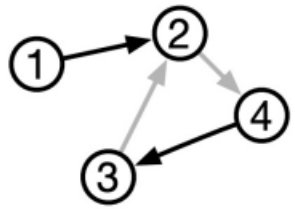
Incoming & outcoming edges for node v_i

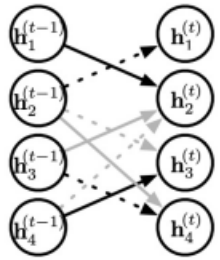
$$\mathbf{h}_i^{(0)} = [\mathbf{x}_i^T, \mathbf{0}]^T$$

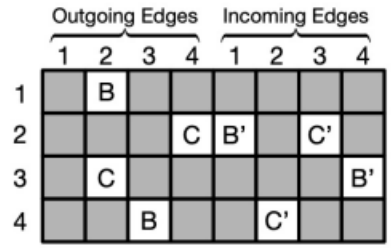
$$\mathbf{a}_i^{(l)} = A_{i:}^T [\mathbf{h}_1^{(l-1)} ... \mathbf{h}_n^{(l-1)}]^T$$

$$\mathbf{h}_i^{(l)} = \mathrm{GRU}(\mathbf{a}_i^{(l)}, \mathbf{h}_i^{(l-1)})$$

GRU for fusing node embeddings



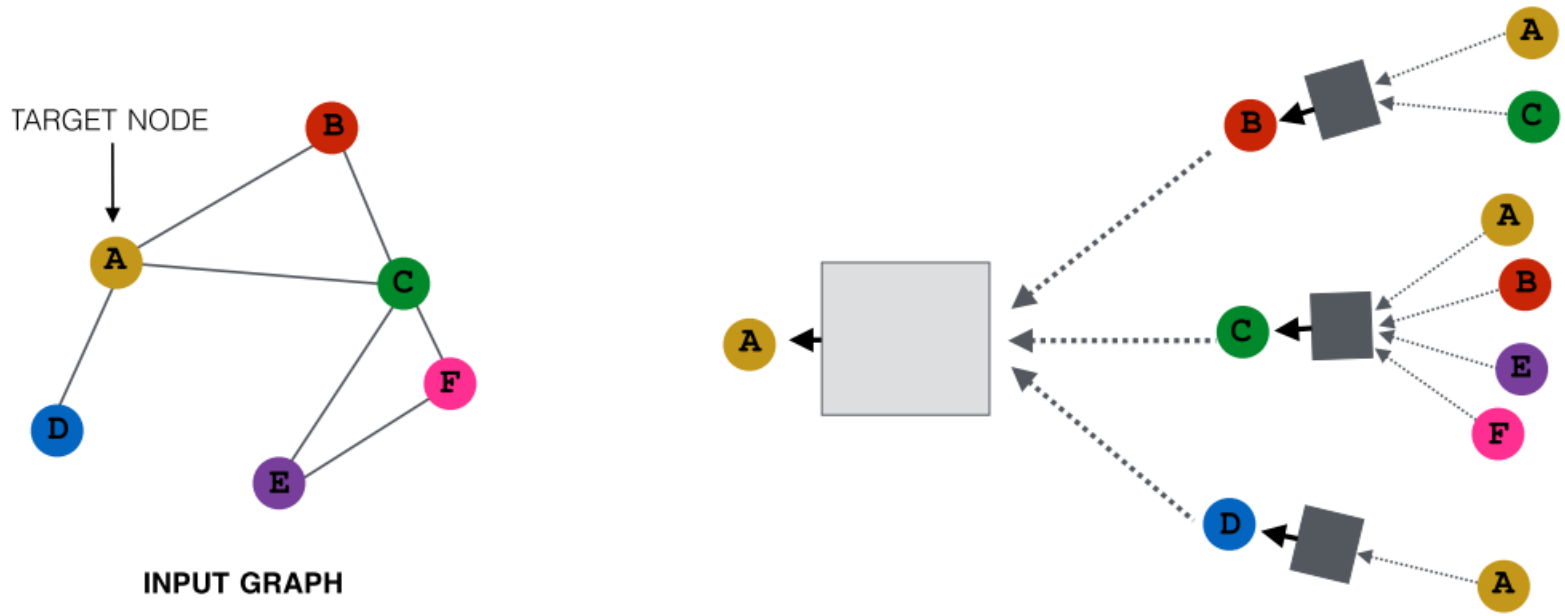




GGNN in NLP Tasks:

- Semantic parsing
- Machine translation

GraphSAGE

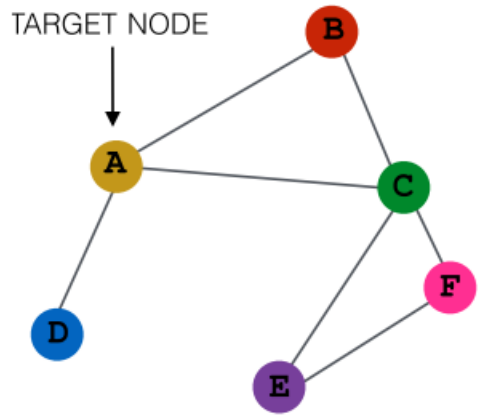


Each node defines a computation graph

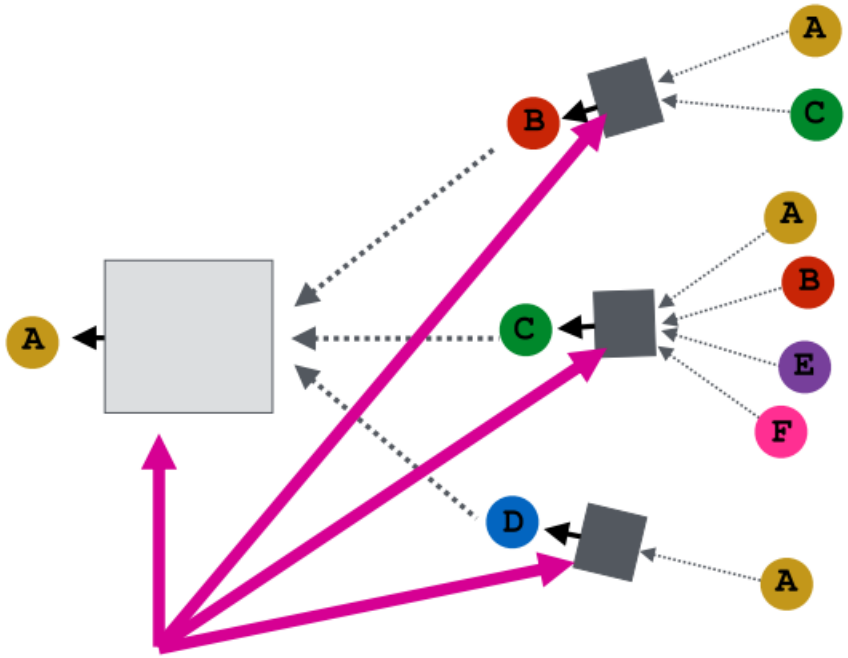
- Each edge in this graph is a transformation/aggregation function

Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

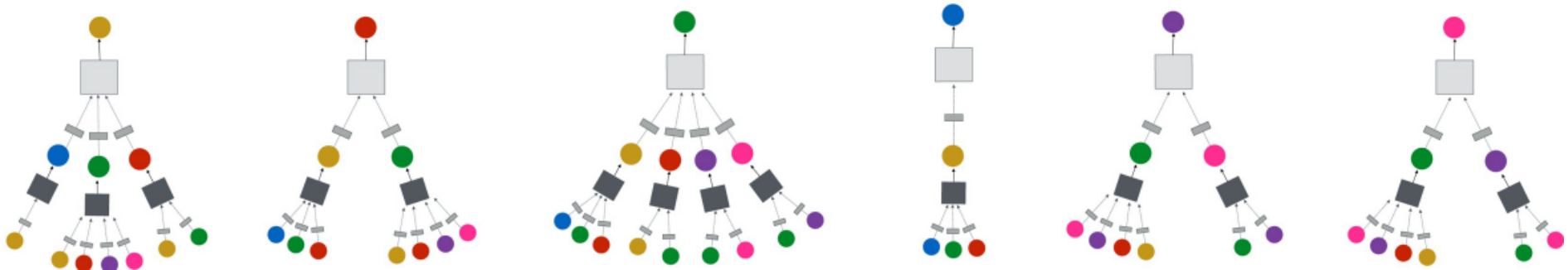
GraphSAGE



INPUT GRAPH



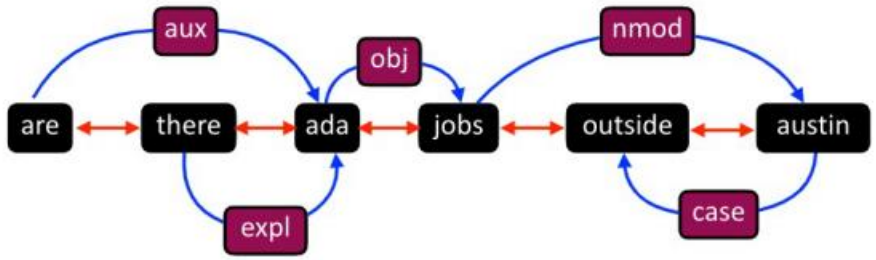
Neural networks



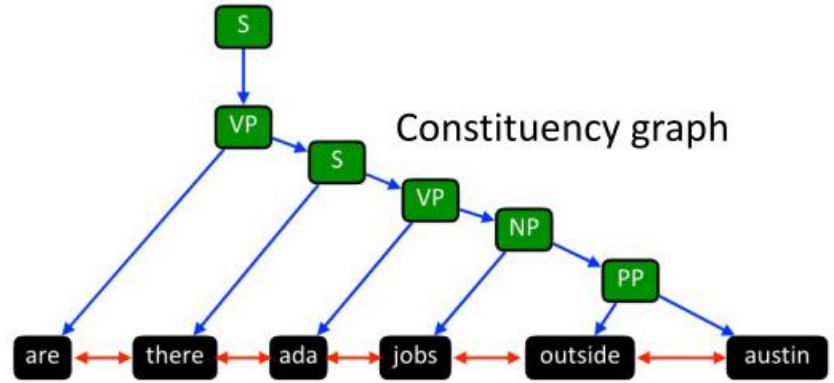
Key Benefits of GraphSAGE

- No manual feature engineering needed
- End-to-end learning results in optimal features.
- Any graph machine learning task:
  - Node-level, link-level, entire graph-level prediction
- Scalable to billion node graphs!

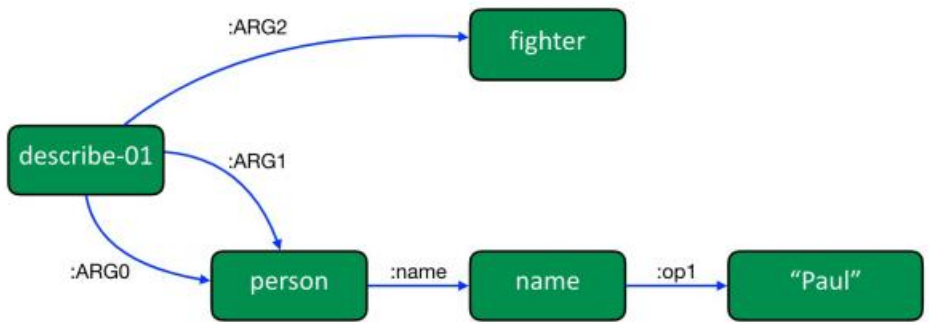
Graphs are ubiquitous in NLP As Well



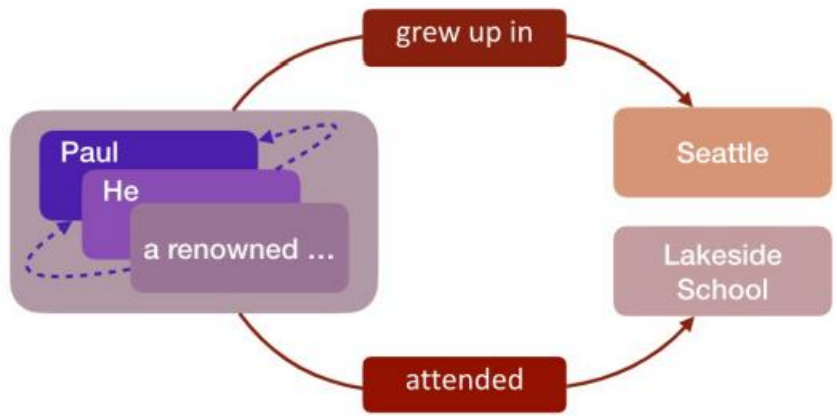
Dependency graph



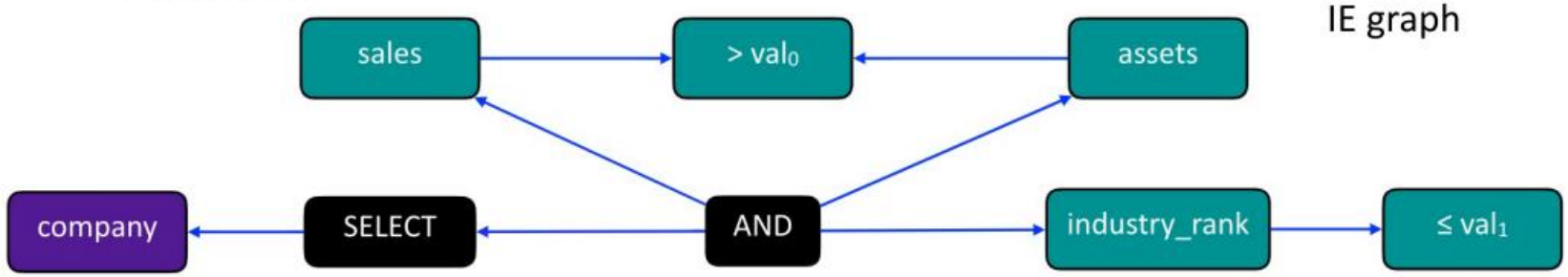
Constituency graph



AMR graph



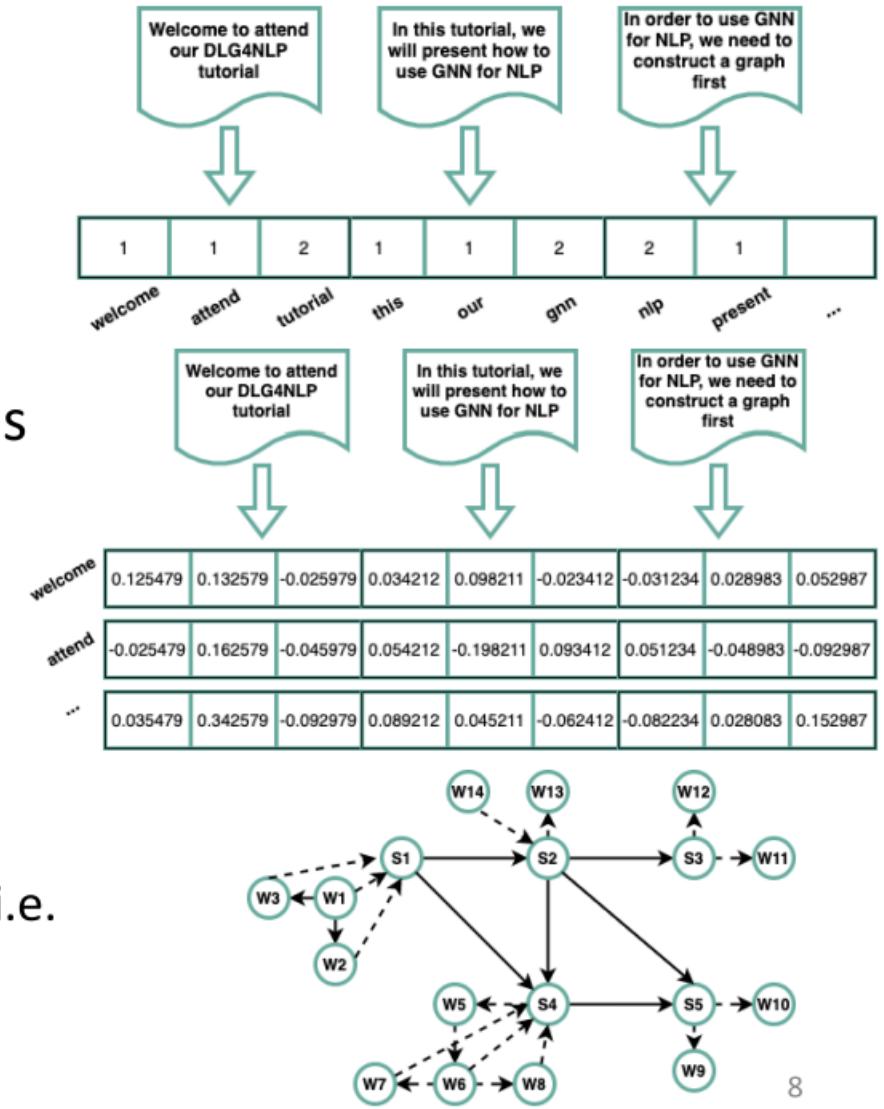
IE graph



SQL graph

Natural Language Processing: A Graph Perspective

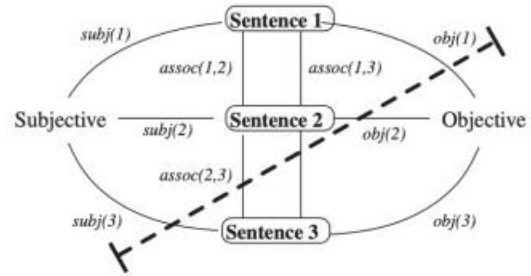
- Represent natural language as a bag of tokens
  - BOW, TF-IDF
  - Topic Modeling: text as a mixture of topics
- Represent natural language as a sequence of tokens
  - Linear-chain CRF
  - Word2vec, Glove
- Represent natural language as a graph
  - Dependency graphs, constituency graphs, AMR graphs, IE graphs, and knowledge graphs
  - Text graph containing multiple hierarchies of elements, i.e. document, sentence and word

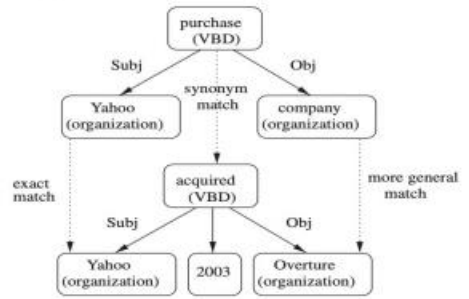


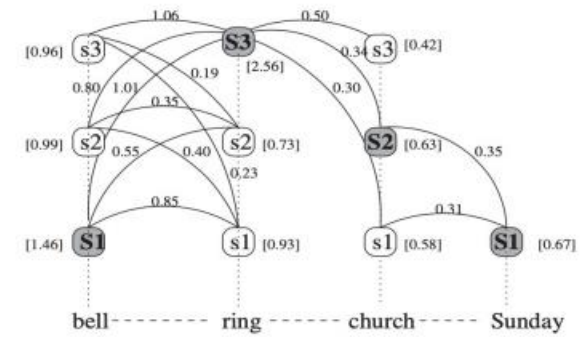
Graph Based Methods for NLP

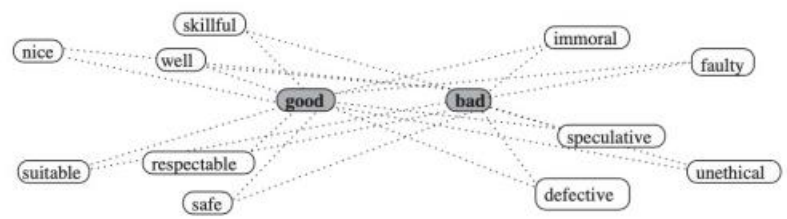
- Random Walk Algorithms
  - Generate random paths, one can obtain a stationary distribution over all the nodes in a graph
  - Applications: semantic similarity of texts, name disambiguation
- Graph Clustering Algorithms
  - Spectral clustering, random walk clustering and min-cut clustering for text clustering
- Graph Matching Algorithms
  - Compute the similarity between two graphs for textual entailment task
- Label Propagation Algorithms
  - Propagate labels from labeled data points to previously unlabeled data points
  - Applications: word-sense disambiguation, sentiment analysis

[Mihalcea and Radev, 2011]



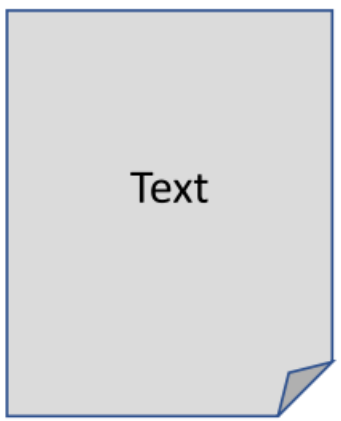


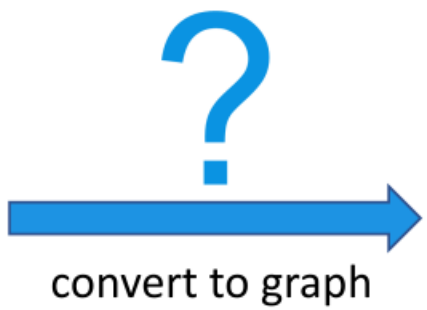


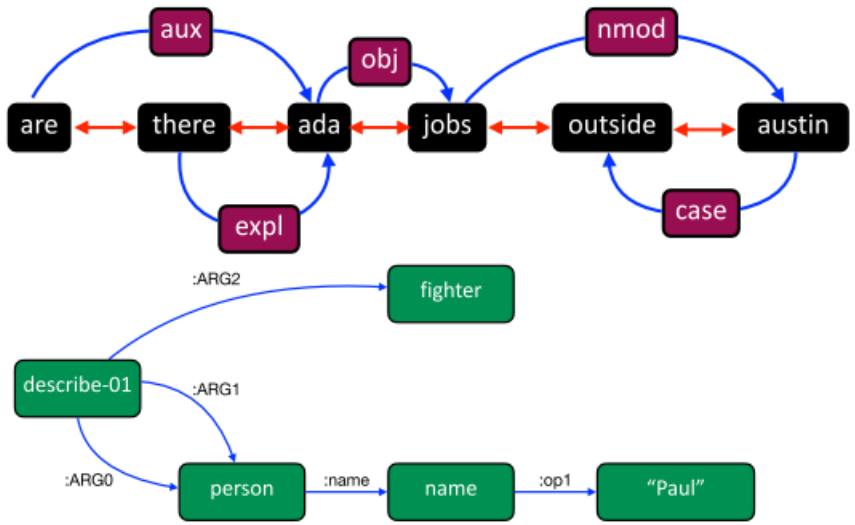


Why Graph Construction for NLP?

- Representation power: graph > sequence > bag
- Different NLP tasks require different aspects of text , e.g., syntax, semantics.
- Different graphs capture different aspects of the text
- Two categories: static vs dynamic graph construction
- Goal: good downstream task performance



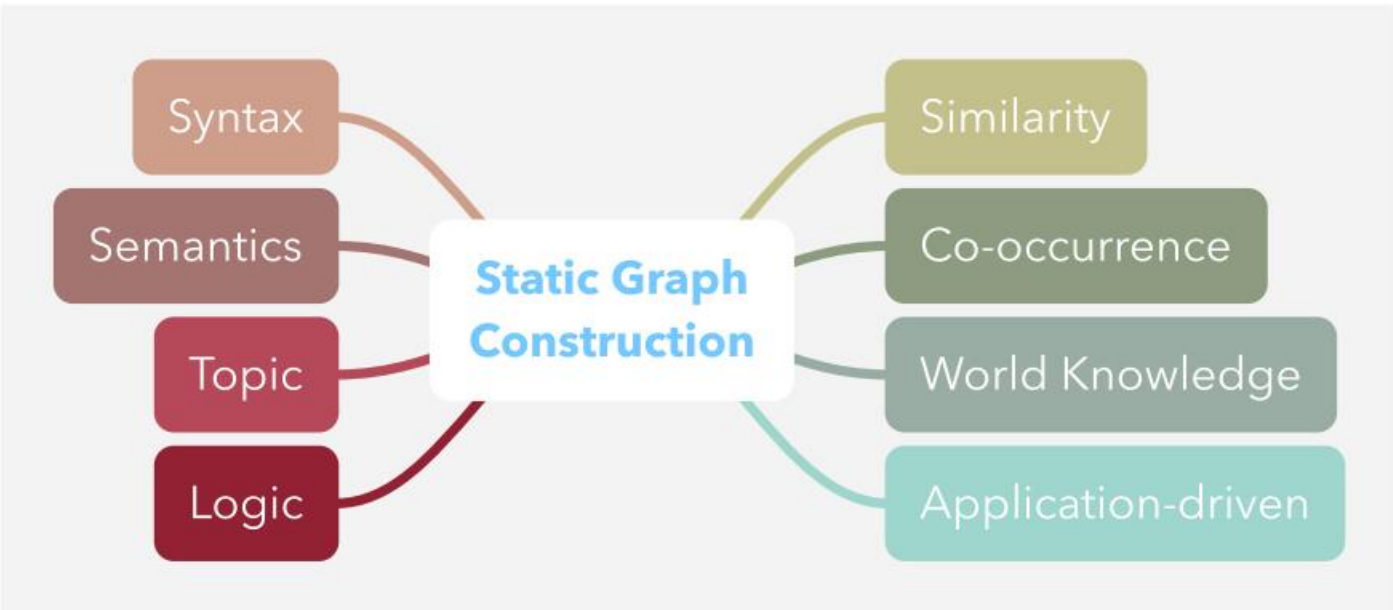




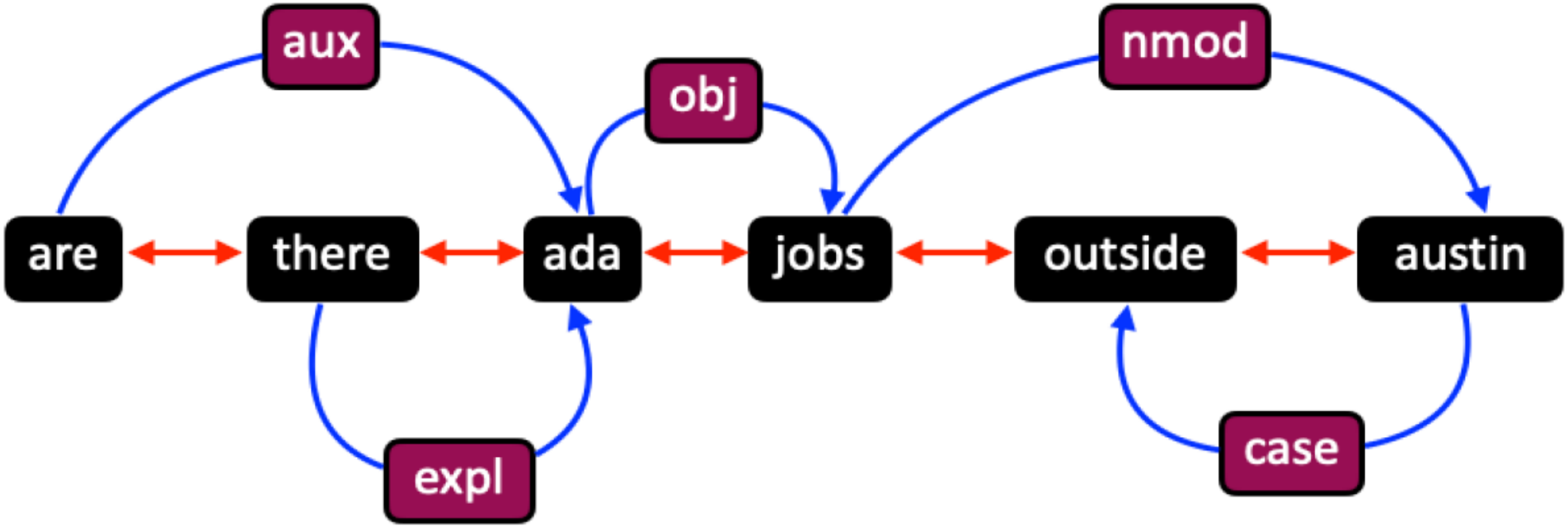
many more graph options…

Static Graph Construction

- Problem setting:
  - Input: raw text (e.g., sentence, paragraph, document, corpus)
  - Output: graph
- Conducted during preprocessing by augmenting text with domain knowledge



Static Graph Construction: Dependency Graph

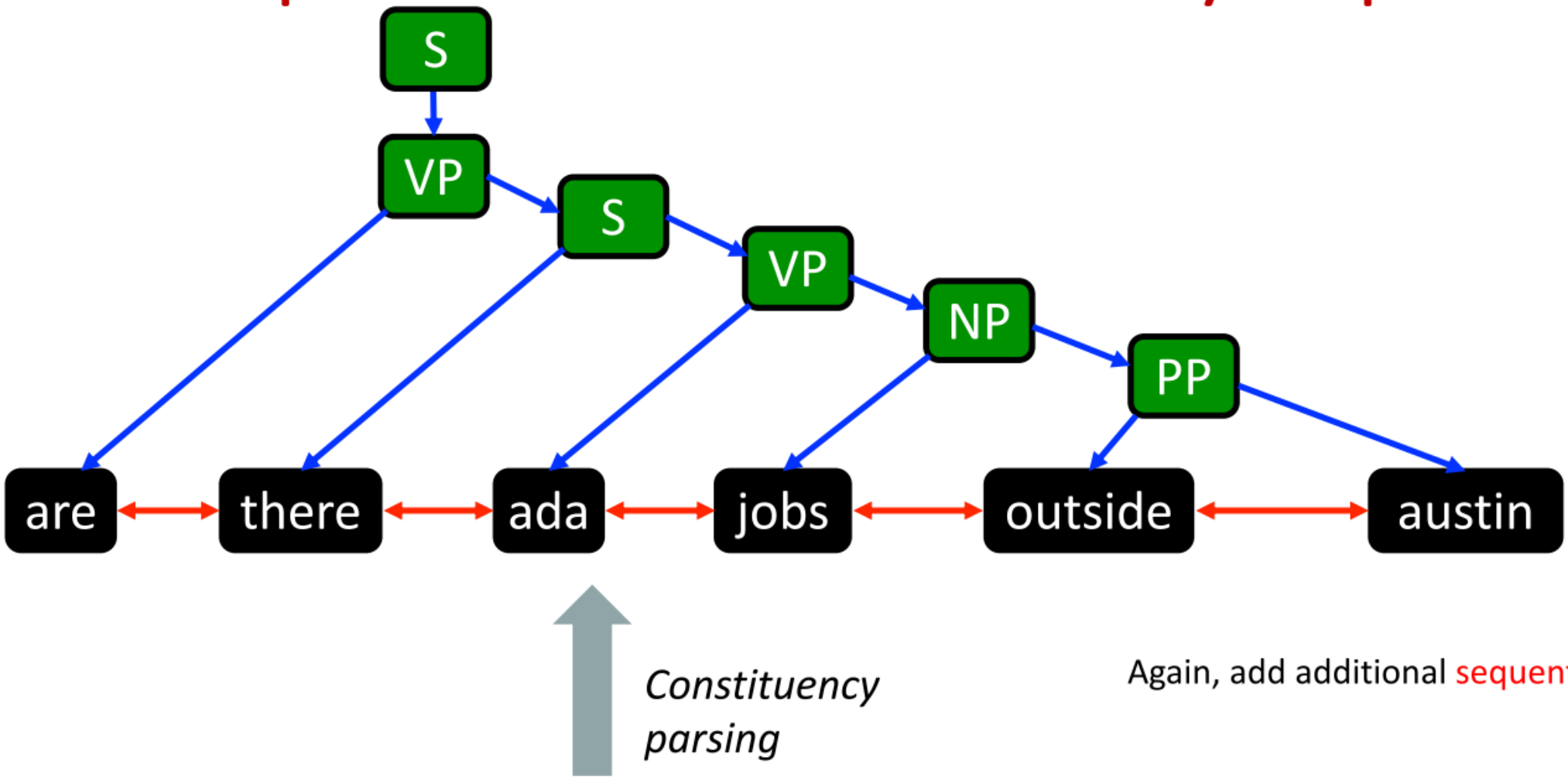


Dependency parsing

Add additional sequential edges to
1) reserve sequential information in raw text
2) connect multiple dependency graphs in a paragraph

Text input: are there ada jobs outside austin

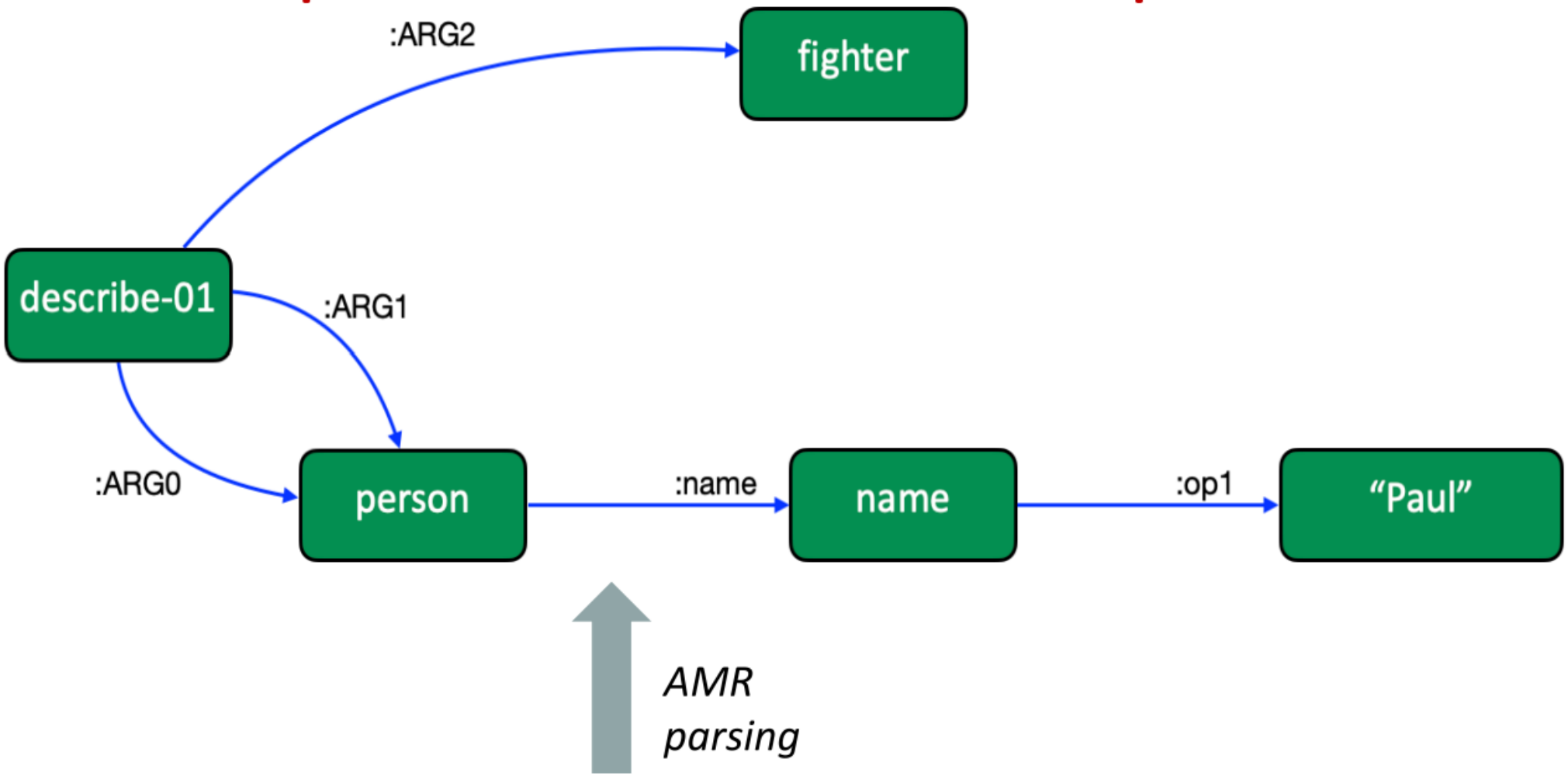
Static Graph Construction: Constituency Graph



Again, add additional sequential edges

Text input: are there ada jobs outside austin

Static Graph Construction: AMR Graph



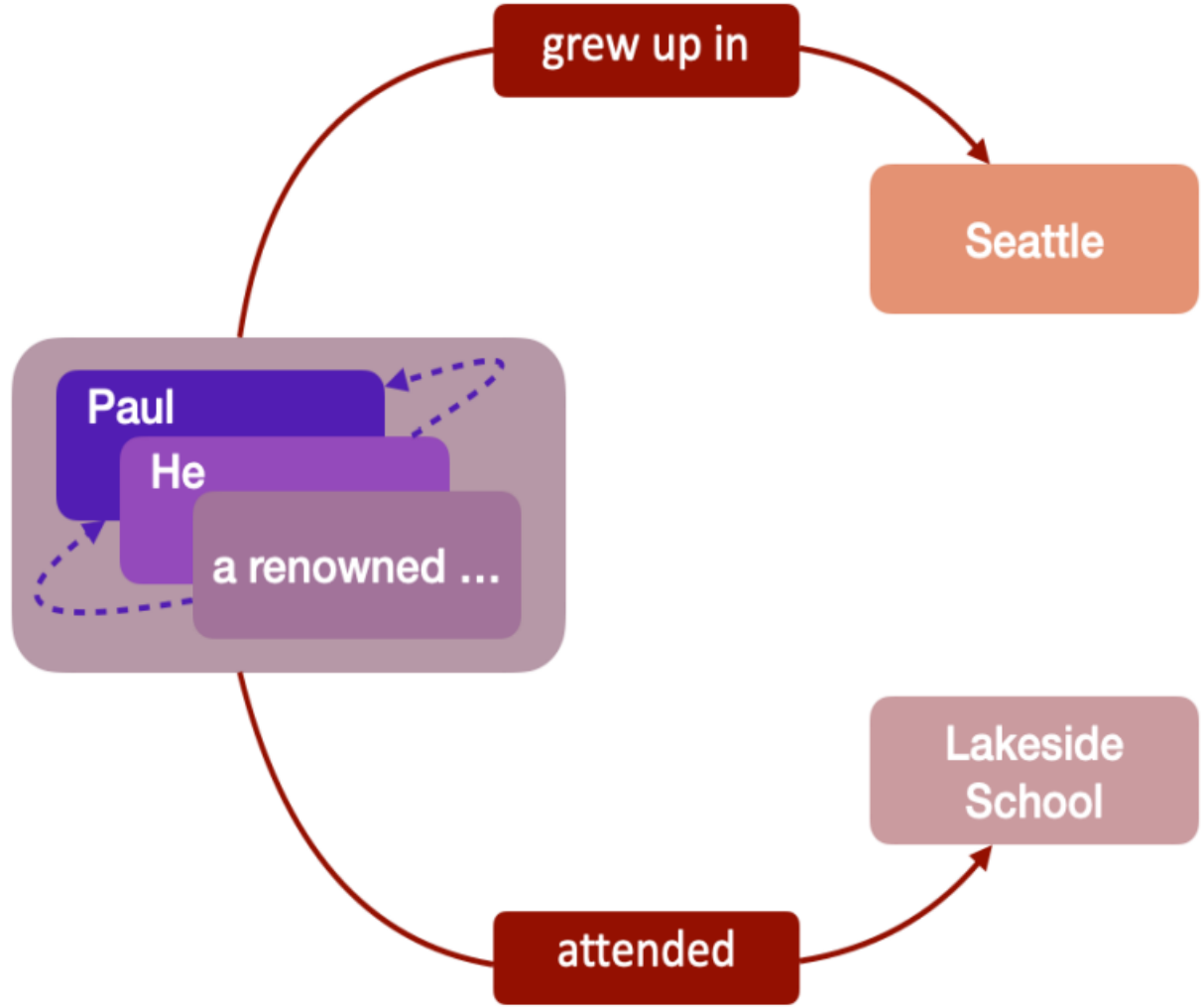
Text input: Paul's description of himself: a fighter

Static Graph Construction: IE Graph

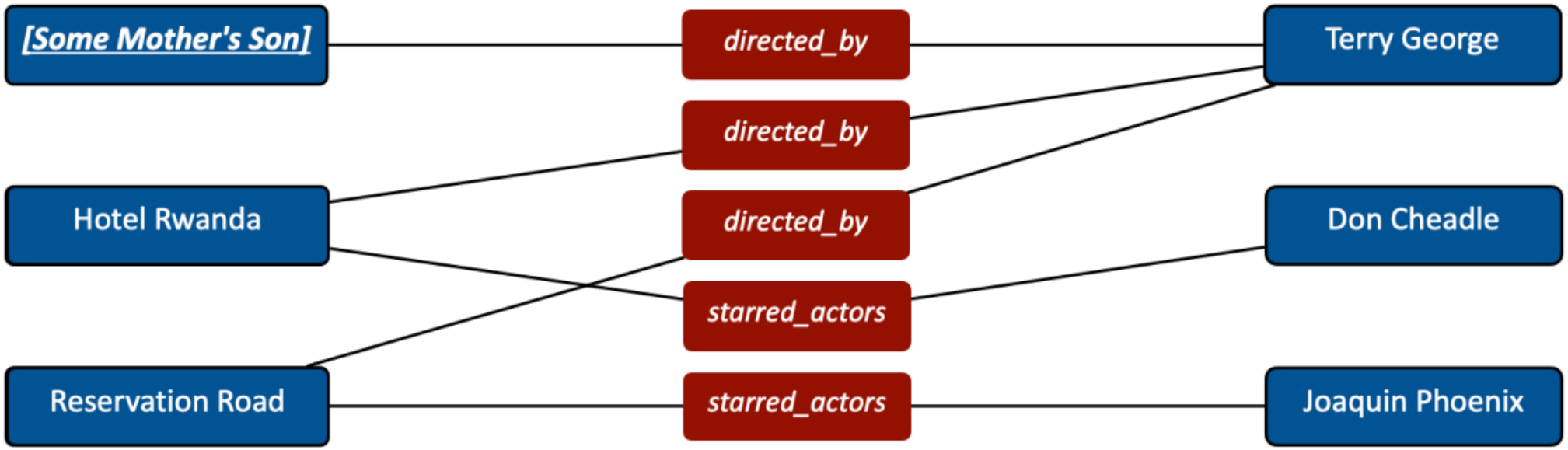
Text input: *Paul, a renowned computer scientist, grew up in Seattle. He attended Lakeside School.*

OpenIE

Coreference



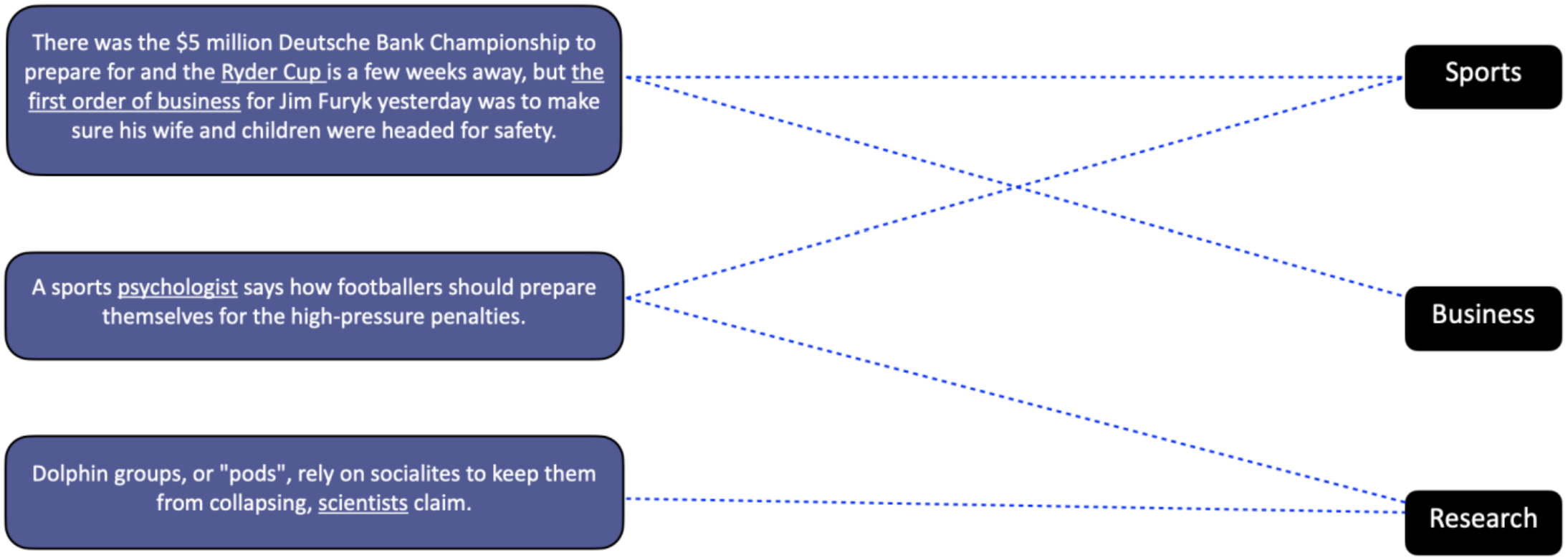
Static Graph Construction: Knowledge Graph



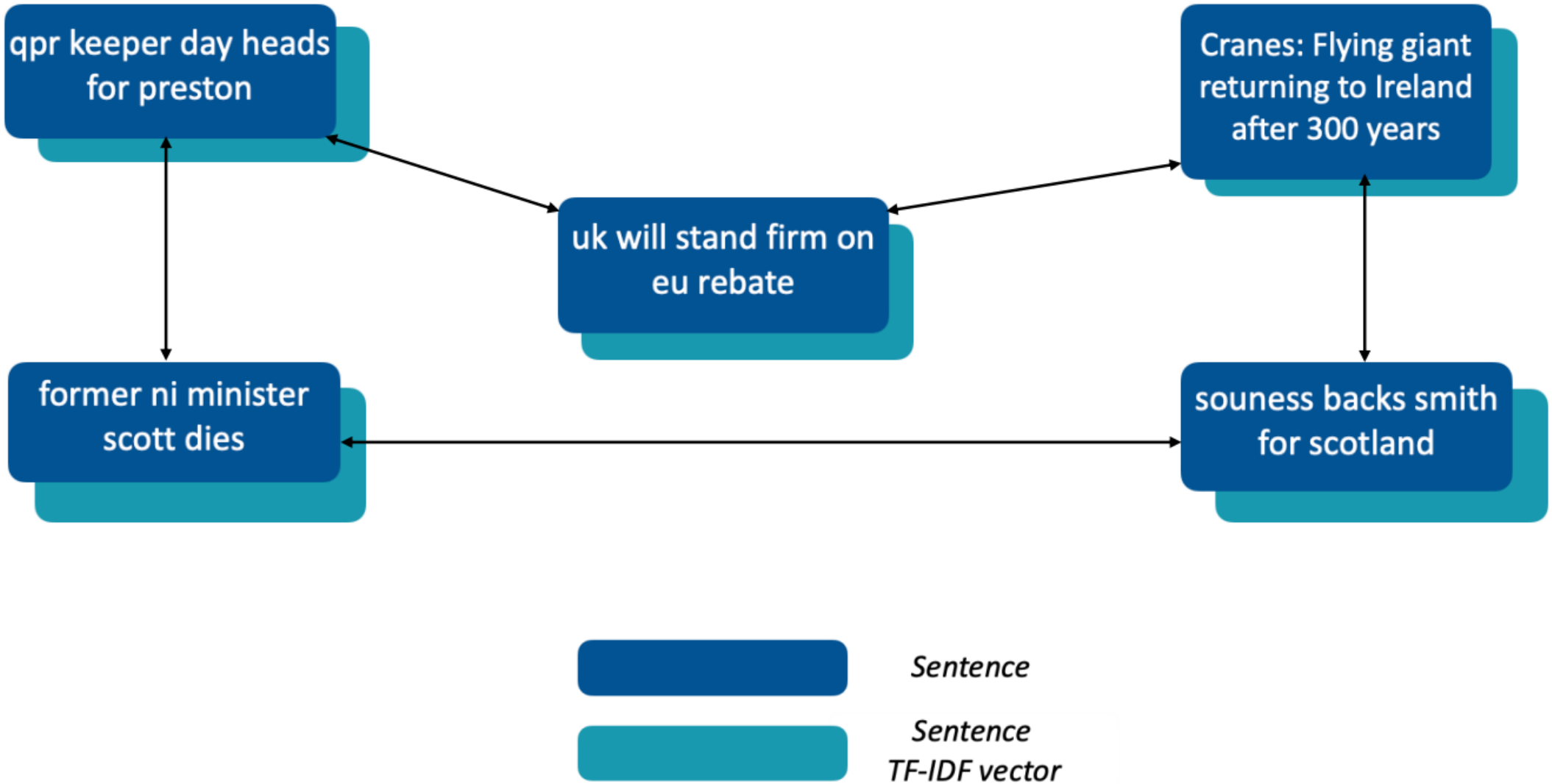
Get the concept sub-graph from KB

Question: *who acted in the movies directed by the director of* ***[Some Mother's Son]***
Answer: *Don Cheadle, Joaquin Phoenix*

Static Graph Construction: Topic Graph



Static Graph Construction: Similarity Graph



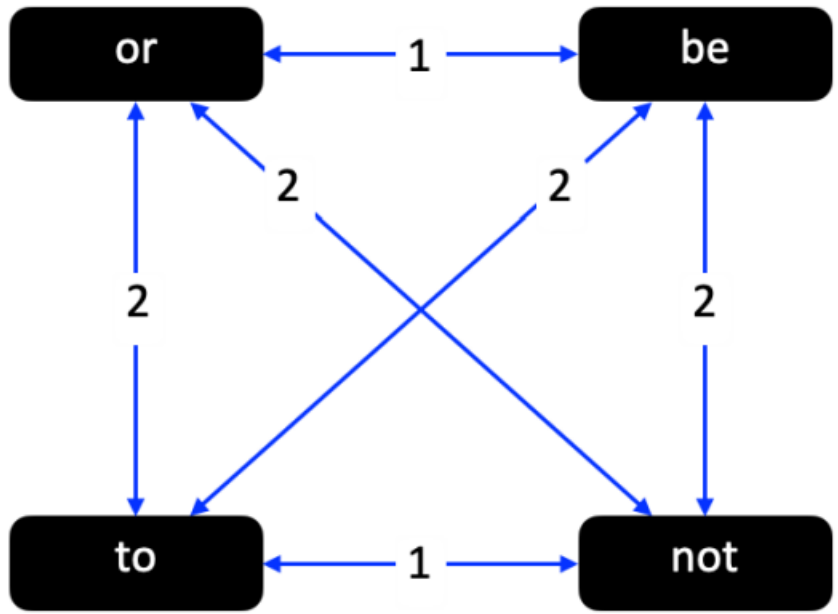
Static Graph Construction: Co-occurrence Graph

Text input: *To be, or not to be: ...*

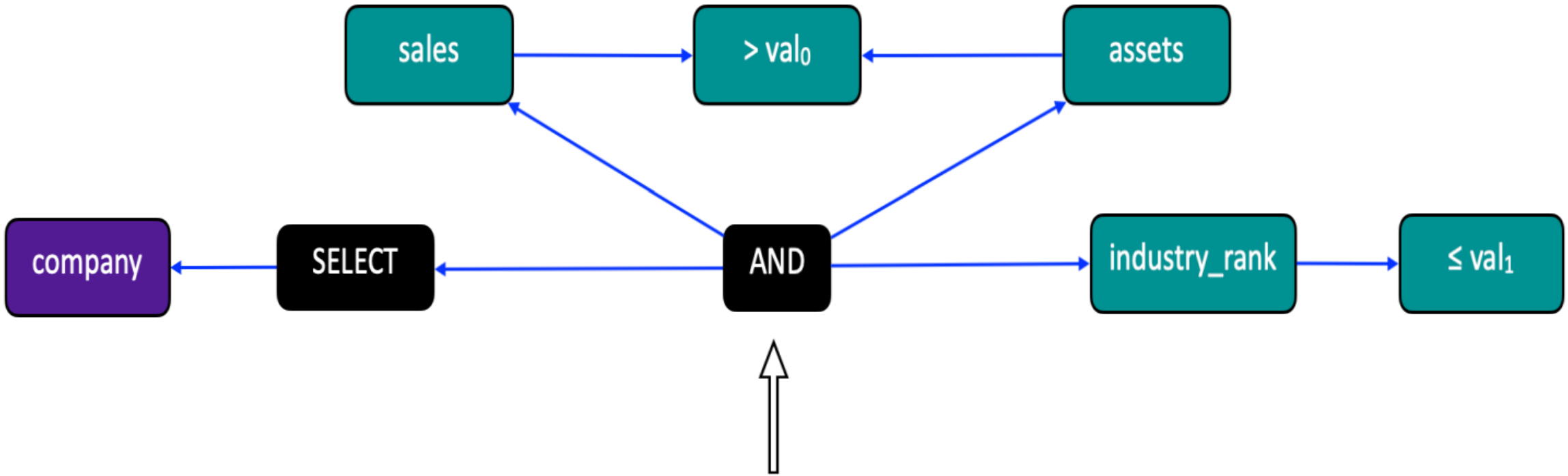
Co-occurrence matrix

	to	be	or	not
to		2	2	1
be	2		1	2
or	2	1		1
not	1	2	1	

Co-occurrence graph

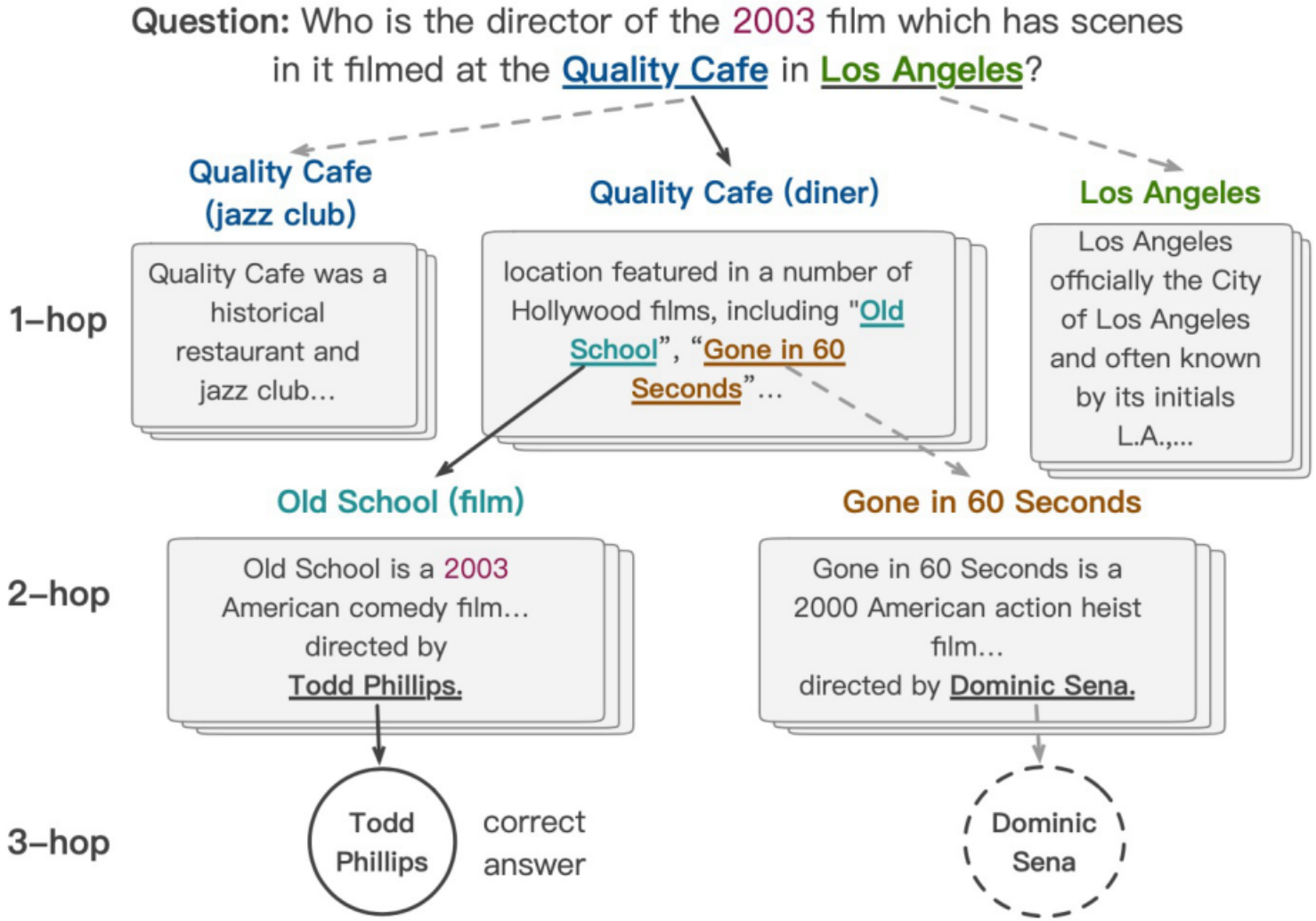


Static Graph Construction: SQL Graph



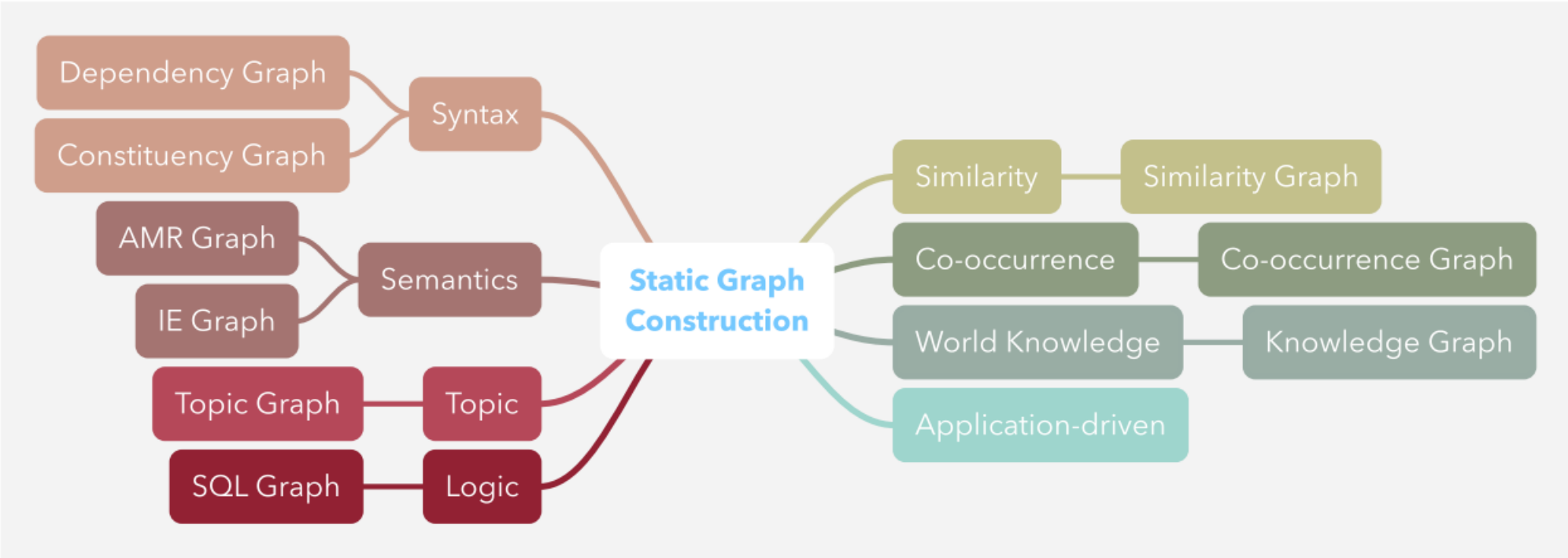
***SQL query input**: **SELECT** company **WHERE** assets > val_0 **AND** sales > val_0 **AND** industry_rank ≤ val_1*

Static Graph Construction: Application-driven Graph



Ming Ding et al. "Cognitive Graph for Multi-Hop Reading Comprehension at Scale". ACL 2019.

Static Graph Construction: Summary

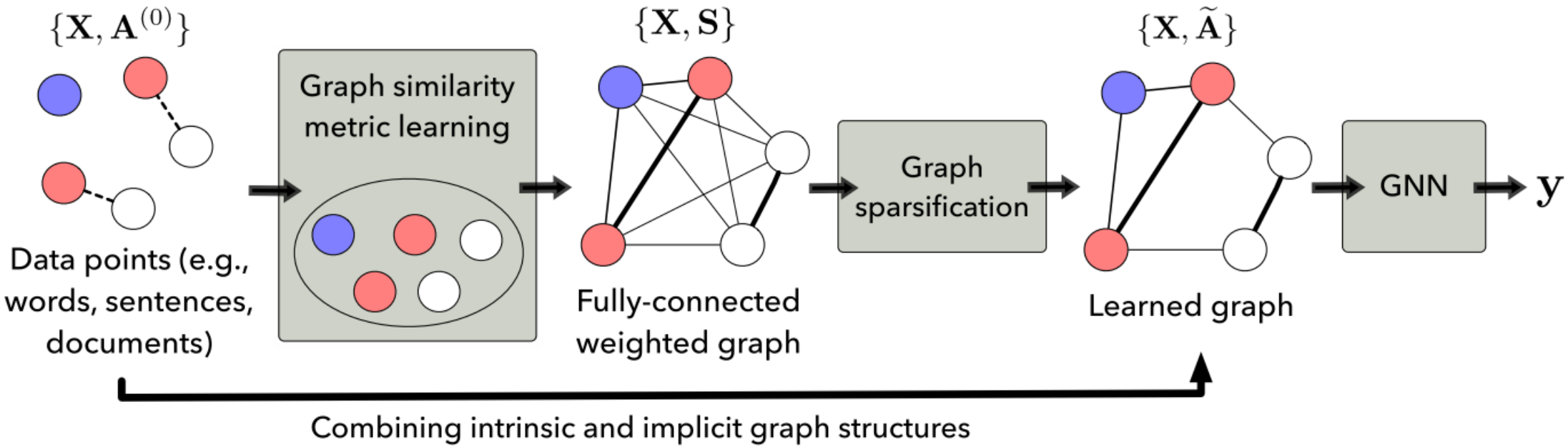


Widely used in various NLP applications such as NLG, MRC, semantic parsing, etc.

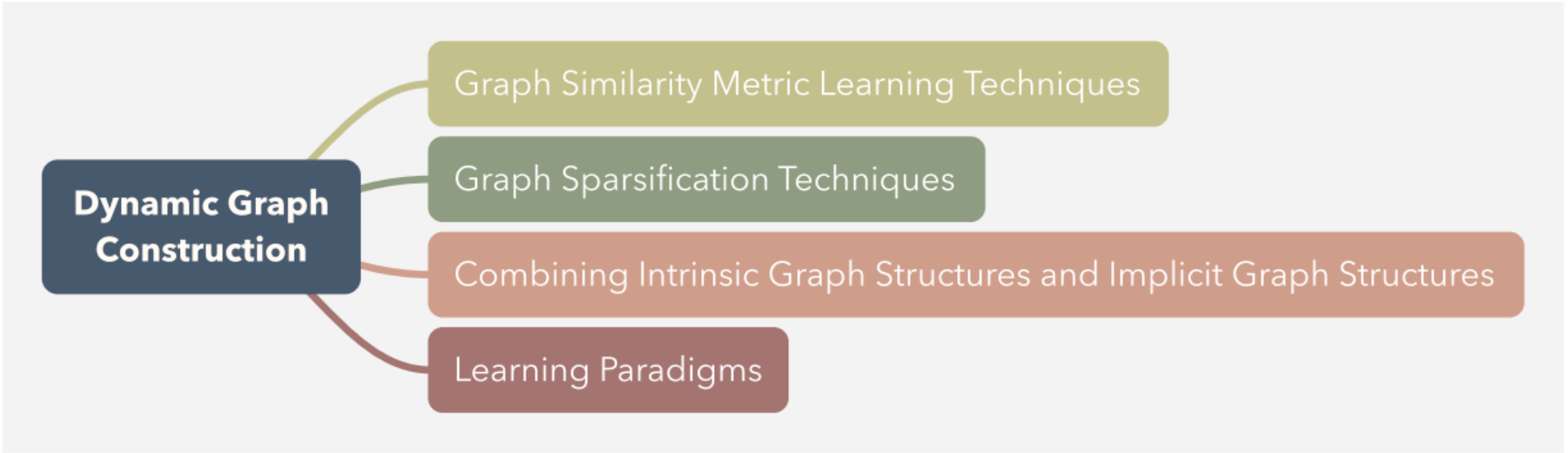
Dynamic Graph Construction

- Problem setting:
  - Input: raw text (e.g., sentence, paragraph, document, corpus)
  - Output: graph
- Graph structure (adjacency matrix) learning on the fly, joint with graph representation learning

Dynamic Graph Construction: Overview



Dynamic Graph Construction Outline



Static vs. Dynamic Graph Construction

New topic in DLG4NLP!

Static graph construction	Dynamic graph construction
Pros	Pros
prior knowledge	no domain expertise
	joint graph structure & representation learning
Cons	Cons
extensive domain expertise	scalability
• error-prone (e.g., noisy, incomplete) • sub-optimal	explainability
• disjoint graph structure & representation learning • error accumulation	

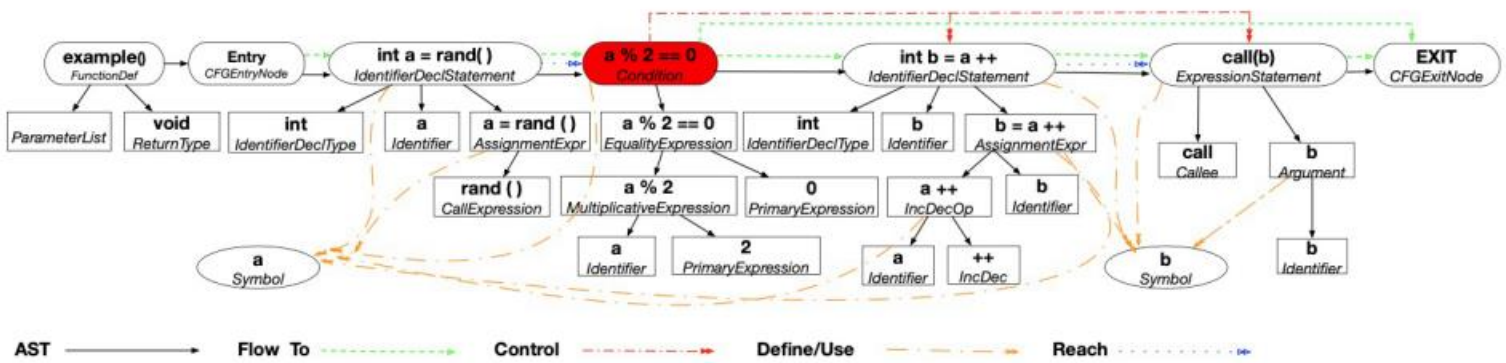
Static vs. Dynamic Graph Construction (cont)

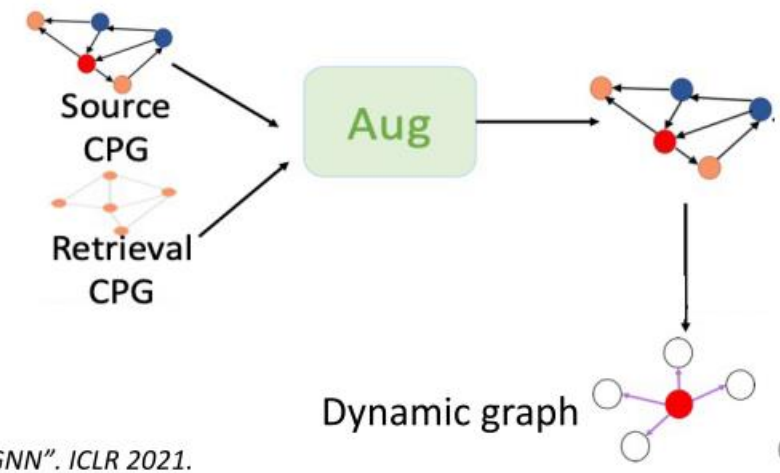
When to use static graph construction

- Domain knowledge which fits the task and can be presented as a graph

When to use dynamic graph construction

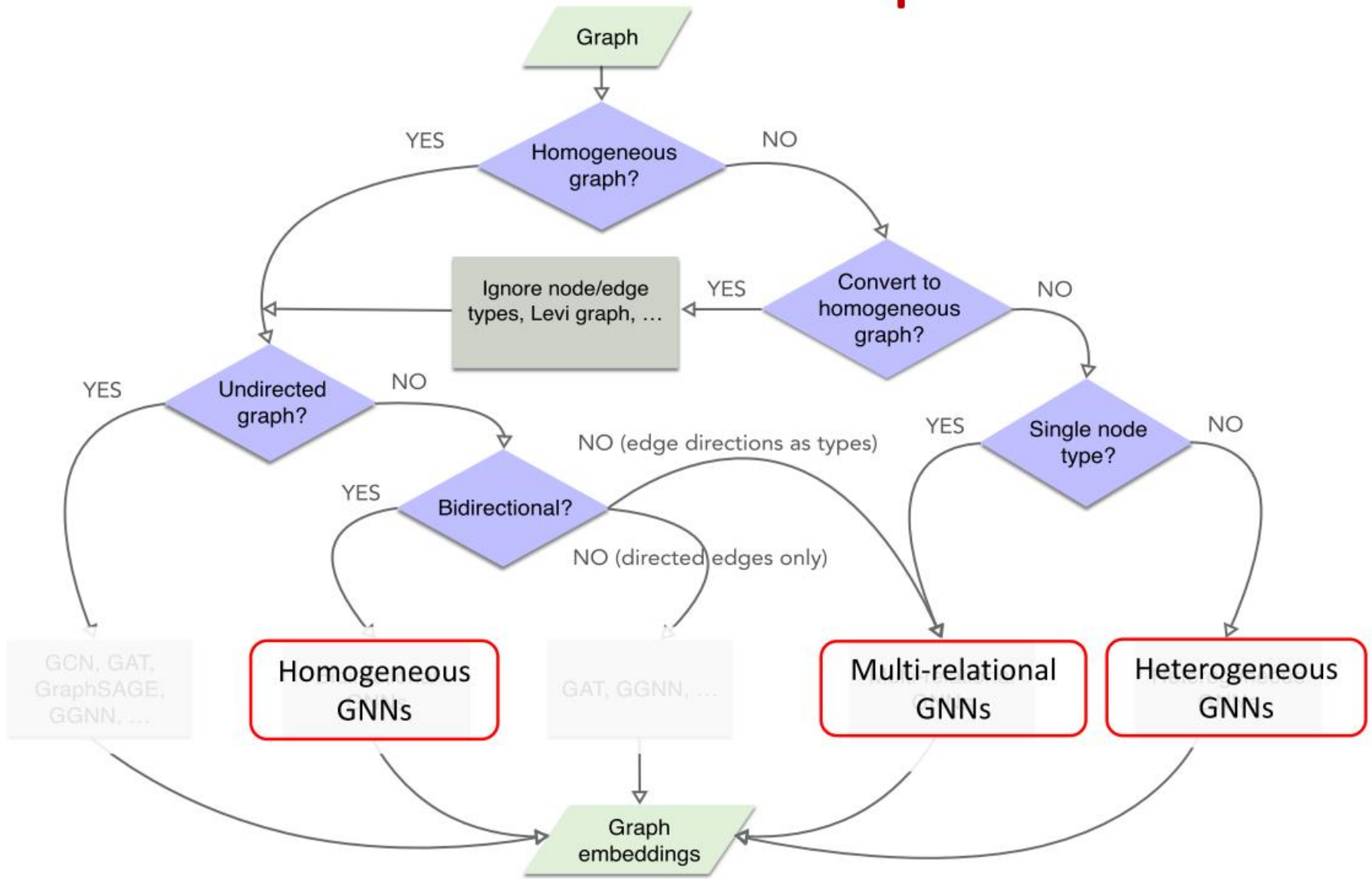
- Lack of domain knowledge which fits the task or can be presented as a graph
- Domain knowledge is incomplete or might contain noise
- To learn implicit graph which augments the static graph



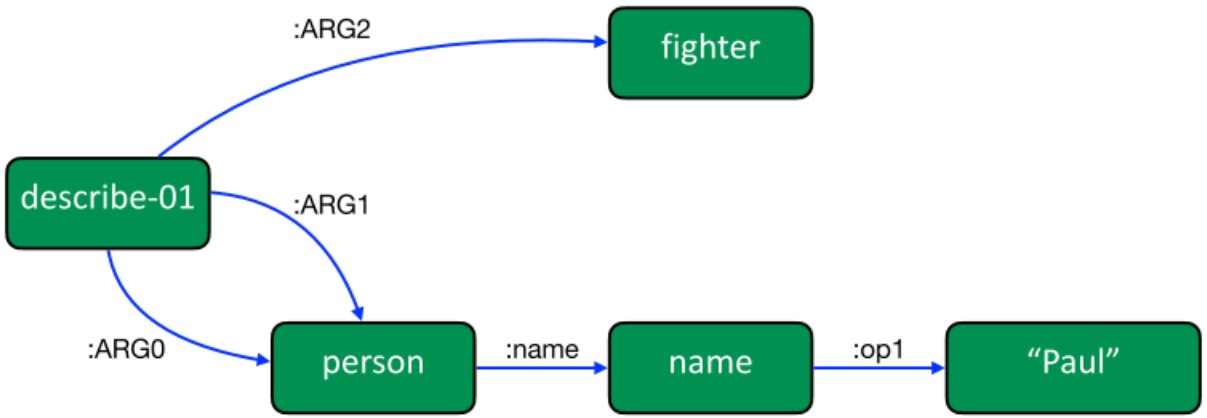


Liu et al. "Retrieval-Augmented Generation for Code Summarization via Hybrid GNN". ICLR 2021.

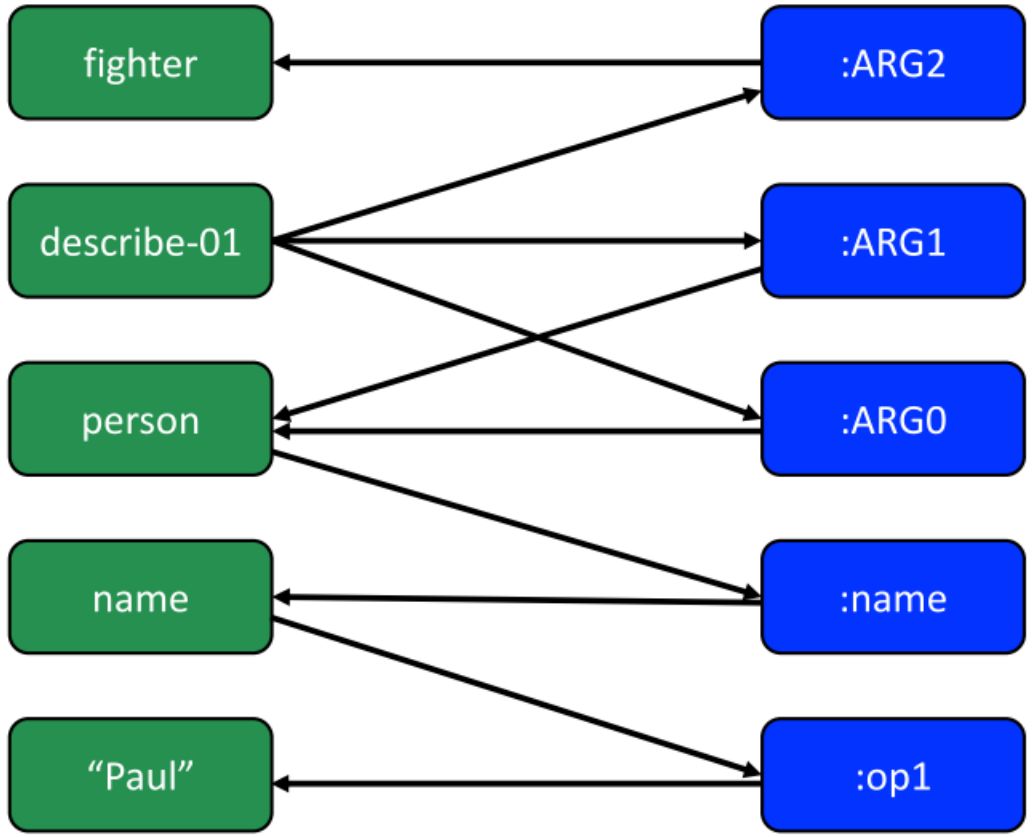
Which GNNs to Use Given a Graph?



Non-homogeneous to Homogeneous Conversion via Levi Graph



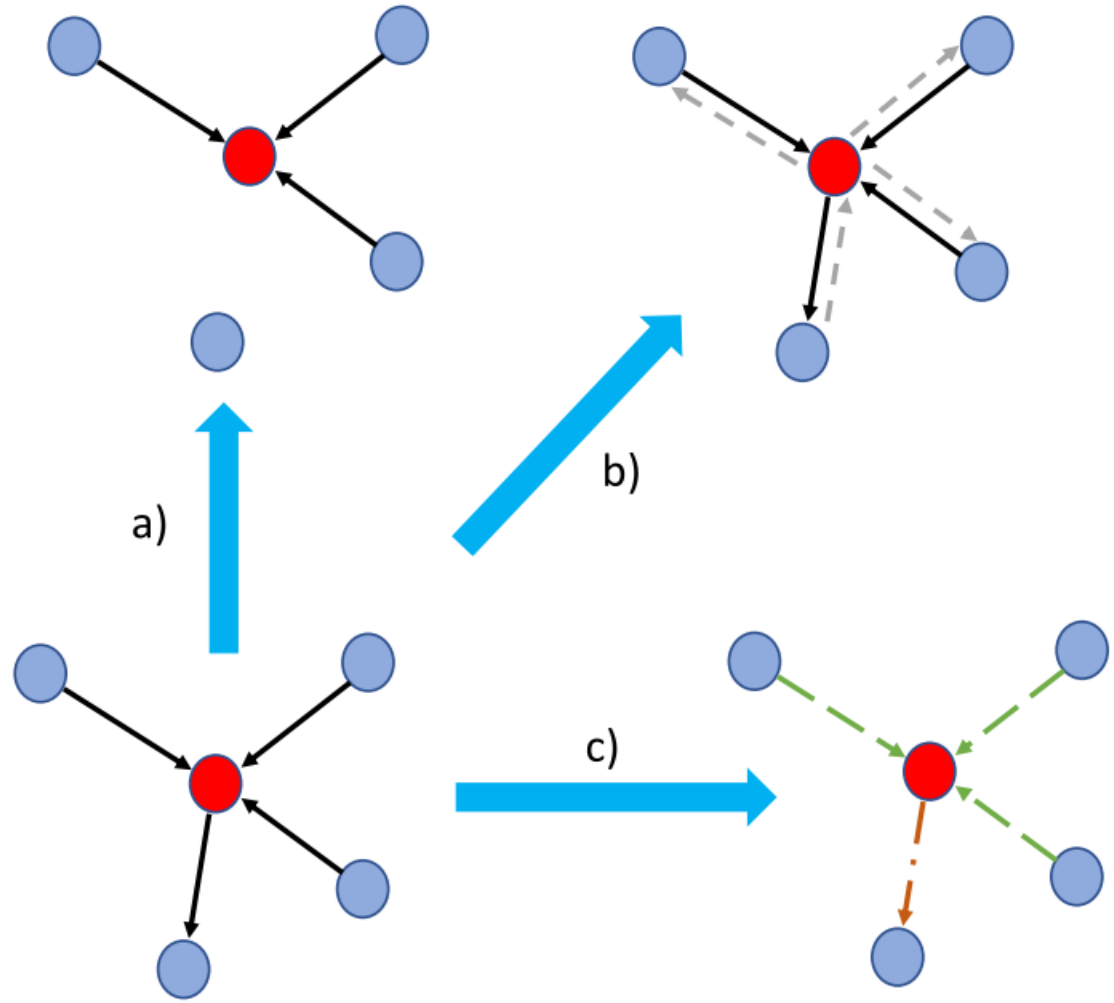
Levi graph conversion



Levi graph: edges as new nodes

How to Handle Edge Direction Information?

- Edge direction is important (think about BiLSTM, BERT)
- Common strategies for handling directed graphs
  a) Message passing only along directed edges (e.g., GAT, GGNN)
  b) Regarding edge directions as edge types (i.e., adding “reverse” edges)
  c) Bidirectional GNNs



Edge Directions as Edge Types

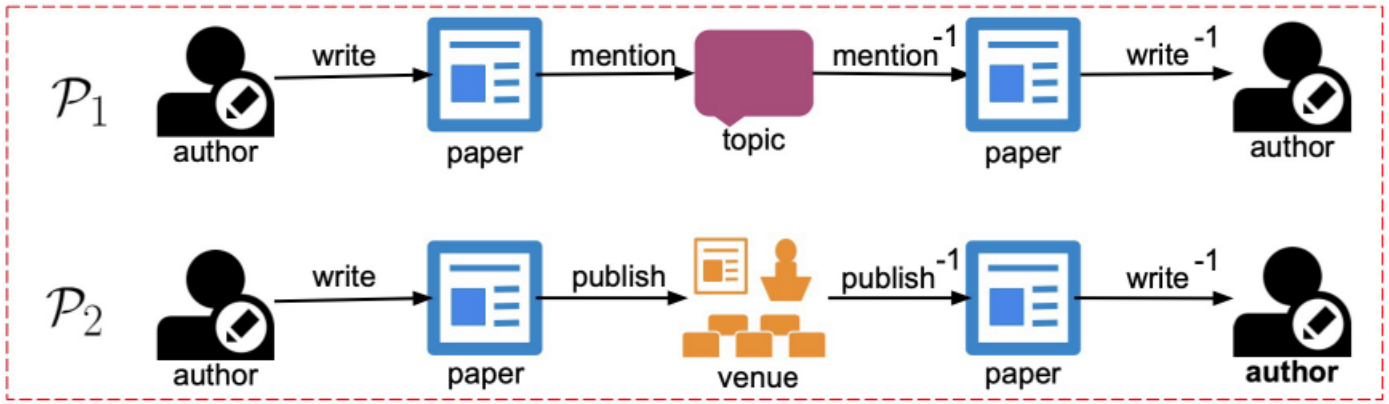
- Regarding edge directions as edge types, resulting in a multi-relational graph

$$dir_{i,j} = \begin{cases} default, & e_{i,j} \text{ is originally existing in the graph} \\ inverse, & e_{i,j} \text{ is the inverse edge} \\ self, & i = j \end{cases}$$

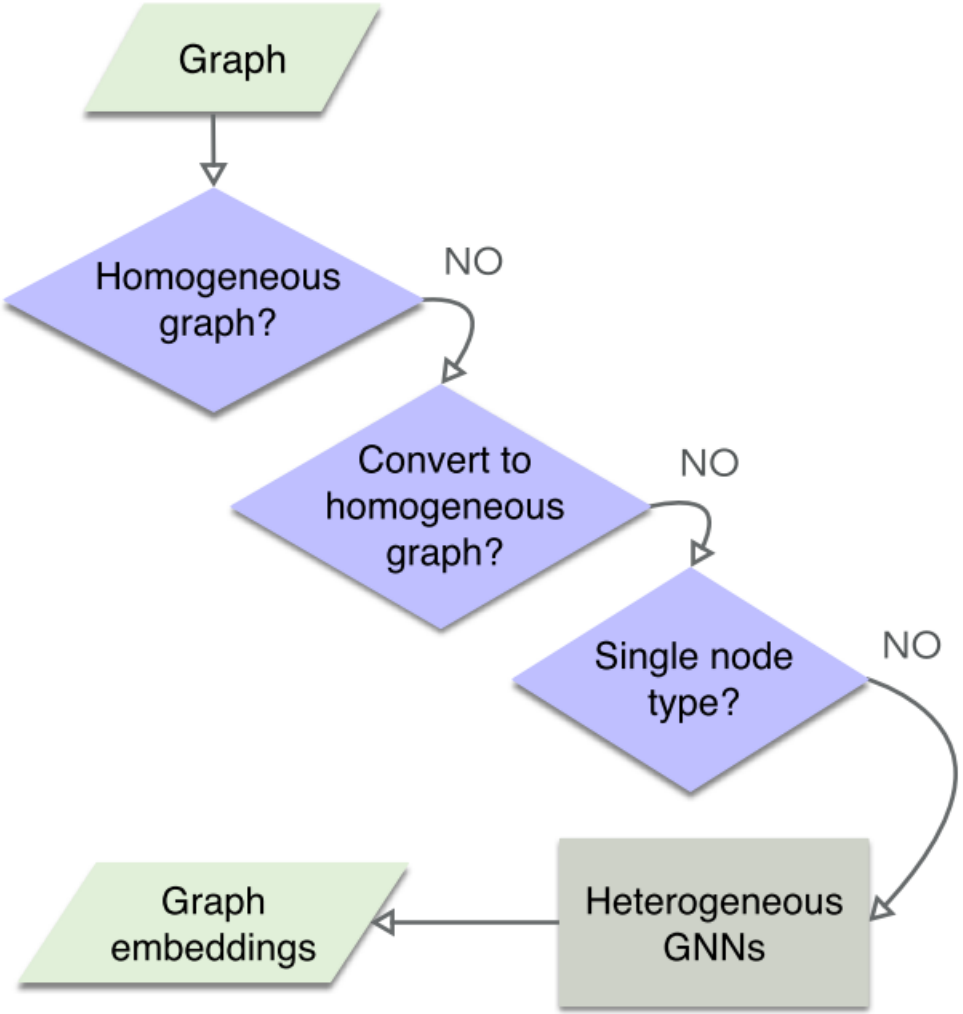
Then we can apply multi-relational GNNs

Heterogeneous GNNs

- When to use Heterogeneous GNNs?
- Heterogeneous GNNs
  a) Meta-path based Heterogeneous GNNs



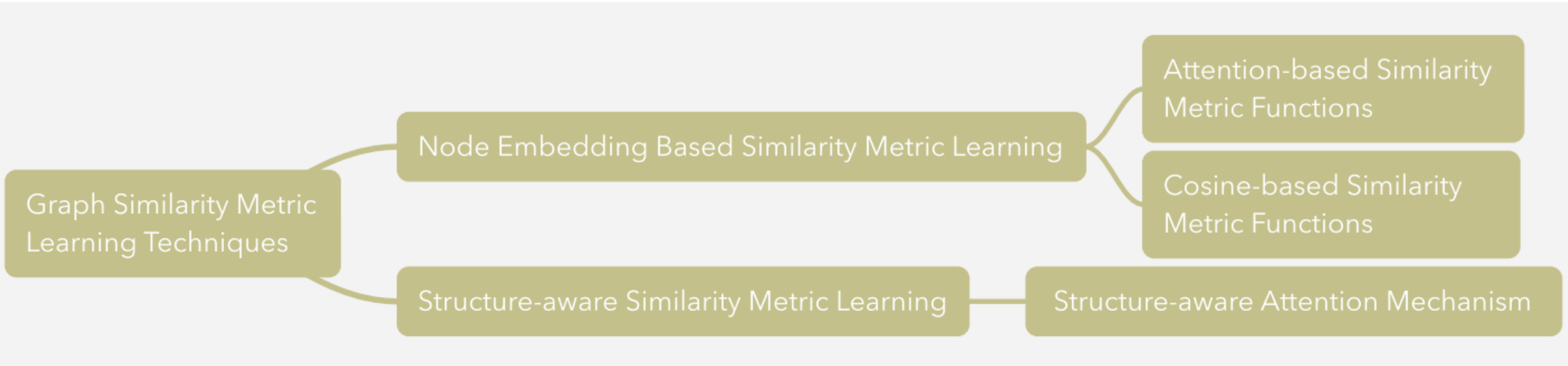
Meta paths among author nodes



Similarity

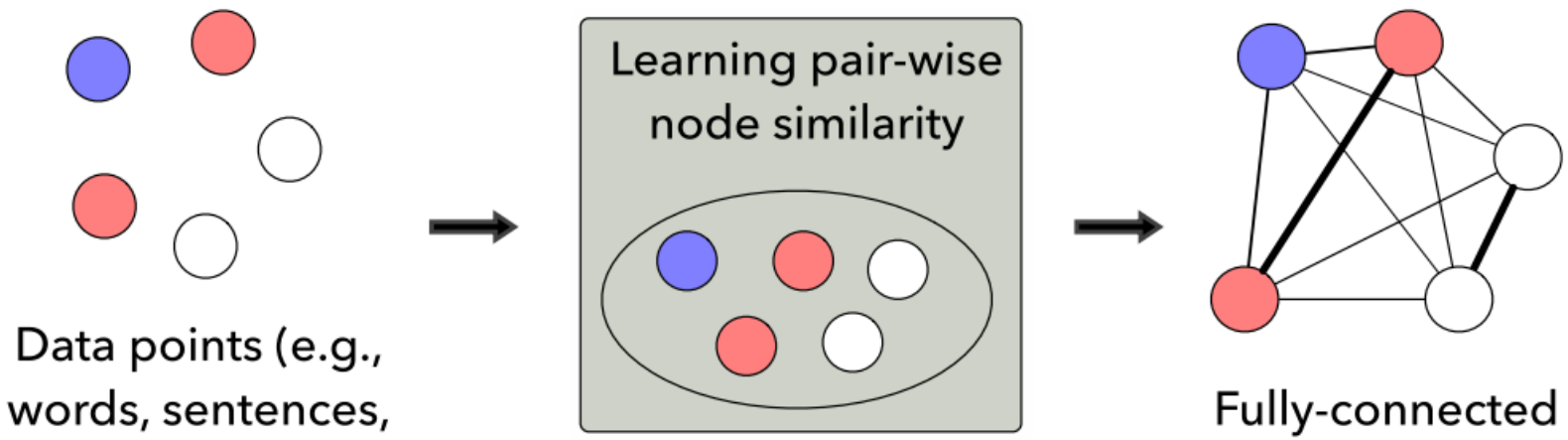
Graph Similarity Metric Learning Techniques

- Graph structure learning as similarity metric learning (in the node embedding space)
- Enabling inductive learning
- Various metric functions



Node Embedding Based Similarity Metric Learning

- Learning a weighted adjacency matrix by computing the pair-wise node similarity in the embedding space
- Common metrics functions
  - Attention-based similarity metric functions
  - Cosine-based similarity metric functions



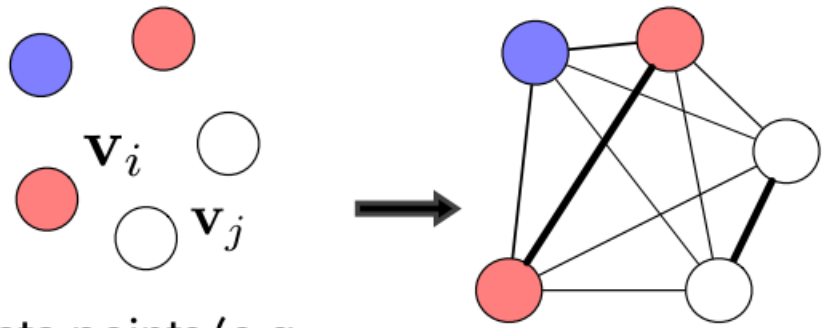
Attention-based Similarity Metric Functions

Variant 1)

$$S_{i,j} = (\mathbf{v}_i \odot \mathbf{u})^T \mathbf{v}_j$$

Node feature vector

Non-negative learnable weight vector



Data points (e.g., words, sentences, documents)

Fully-connected weighted graph

Variant 2)

$$S_{i,j} = \mathrm{ReLU}(\mathbf{W}\mathbf{v}_i)^T \mathrm{ReLU}(\mathbf{W}\mathbf{v}_j)$$

Learnable weight matrix

Chen at al. "GraphFlow: Exploiting Conversation Flow with Graph Neural Networks for Conversational Machine Comprehension". IJCAI 2020.

Chen et al. "Reinforcement Learning Based Graph-to-Sequence Model for Natural Question Generation". ICLR 2020.

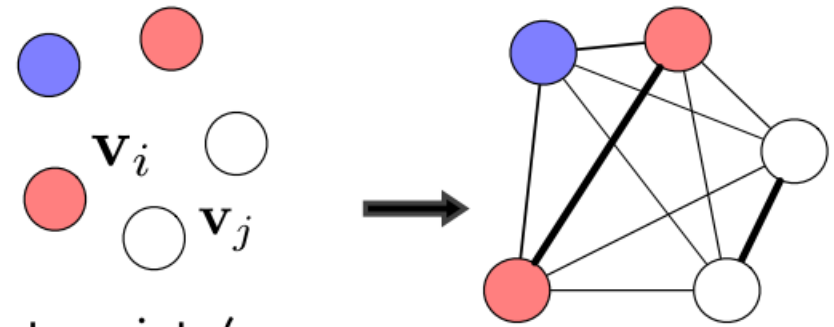
Cosine-based Similarity Metric Functions

$$S_{i,j}^{p} = \cos\left(\mathbf{w}_p \odot \mathbf{v}_i, \mathbf{w}_p \odot \mathbf{v}_j\right)$$

Learnable weight vector

$$S_{i,j} = \frac{1}{m}\sum_{p=1}^{m} S_{ij}^{p}$$

Multi-head similarity scores

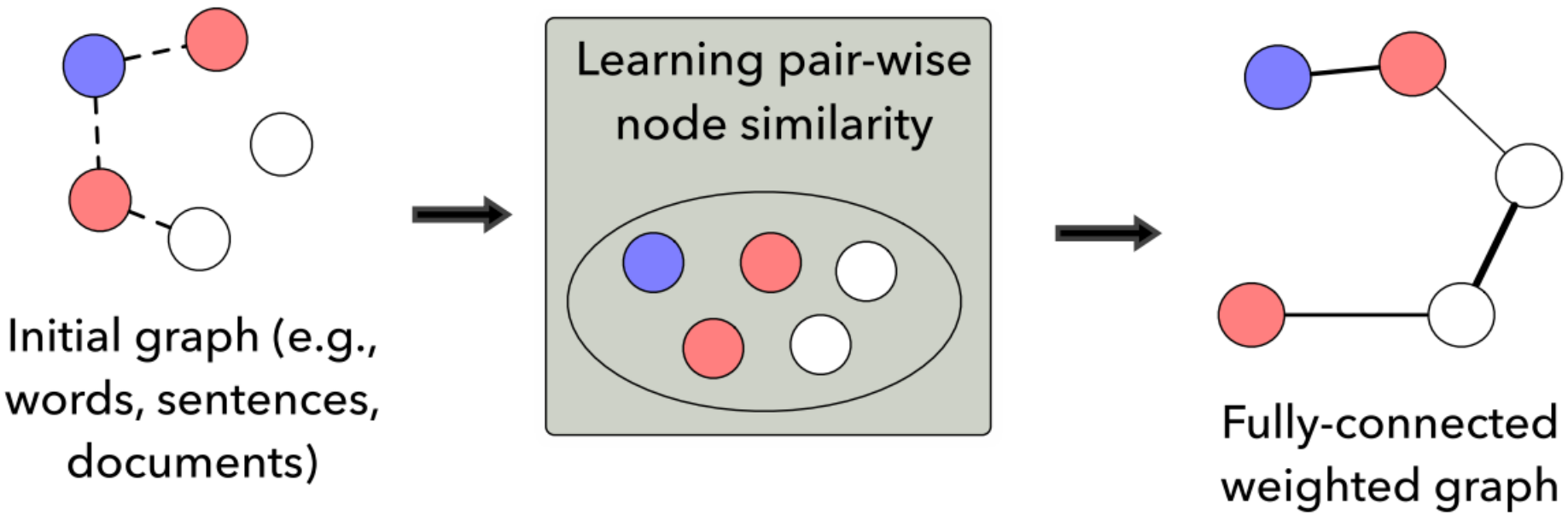


Data points (e.g., words, sentences, documents)

Fully-connected weighted graph

Structure-aware Similarity Metric Learning

- Learning a weighted adjacency matrix by computing the pair-wise node similarity in the embedding space
- Considering existing edge information of the intrinsic graph in addition to the node information



Attention-based Similarity Metric Functions

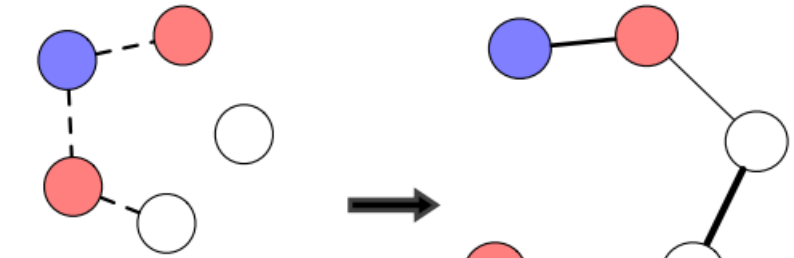
Variant 1)

$$S_{i,j}^{l} = \text{softmax}(\mathbf{u}^T \tanh(\mathbf{W}[\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{i,j}]))$$

Edge embeddings

Variant 2)

$$S_{i,j} = \frac{\text{ReLU}(\mathbf{W}^Q \mathbf{v}_i)^T (\text{ReLU}(\mathbf{W}^K \mathbf{v}_i) + \text{ReLU}(\mathbf{W}^R \mathbf{e}_{i,j}))}{\sqrt{d}}$$



Initial graph (e.g., words, sentences, documents)

Fully-connected weighted graph

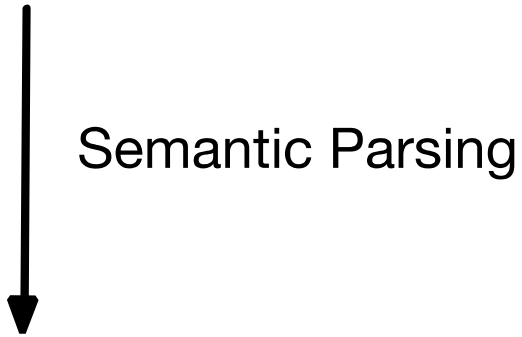
Liu et al. "Contextualized Non-local Neural Networks for Sequence Learning". AAAI 2019.

Liu et al. "Retrieval-Augmented Generation for Code Summarization via Hybrid GNN". ICLR 2021.

Parsing

Semantic Parsing Task

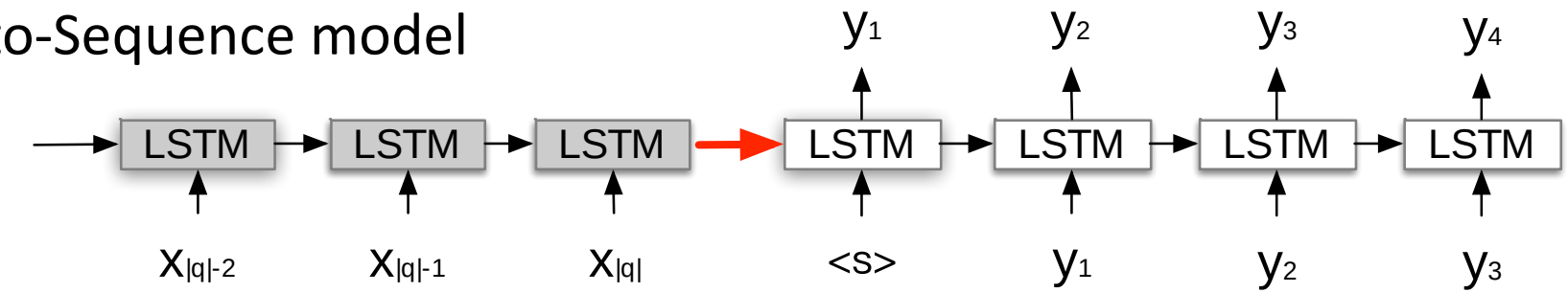
What are the jobs for programmer that has salary 50000 that uses c++ and not related with AI



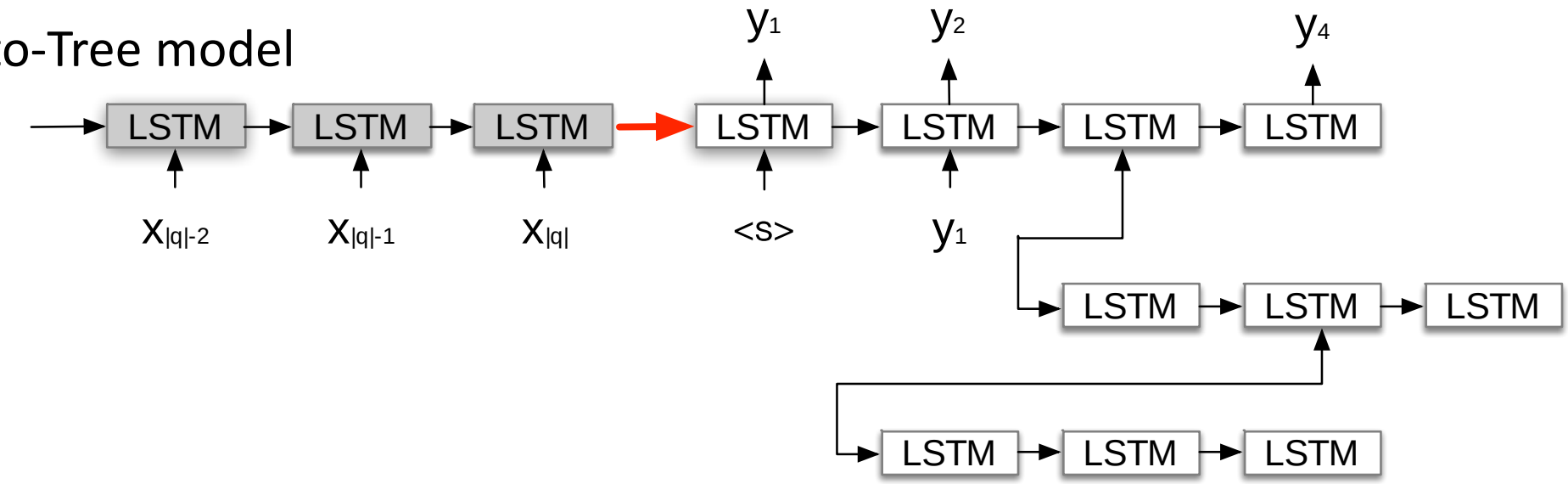
answer(J, (job(J), -((area(J,R), const(R, 'ai'))), language(J, L), const(P, 'Programmer'), title(J, P), const(P, 'Programmer'), salary_greater_than(J, 50000, year))))

Neural Semantic Parser

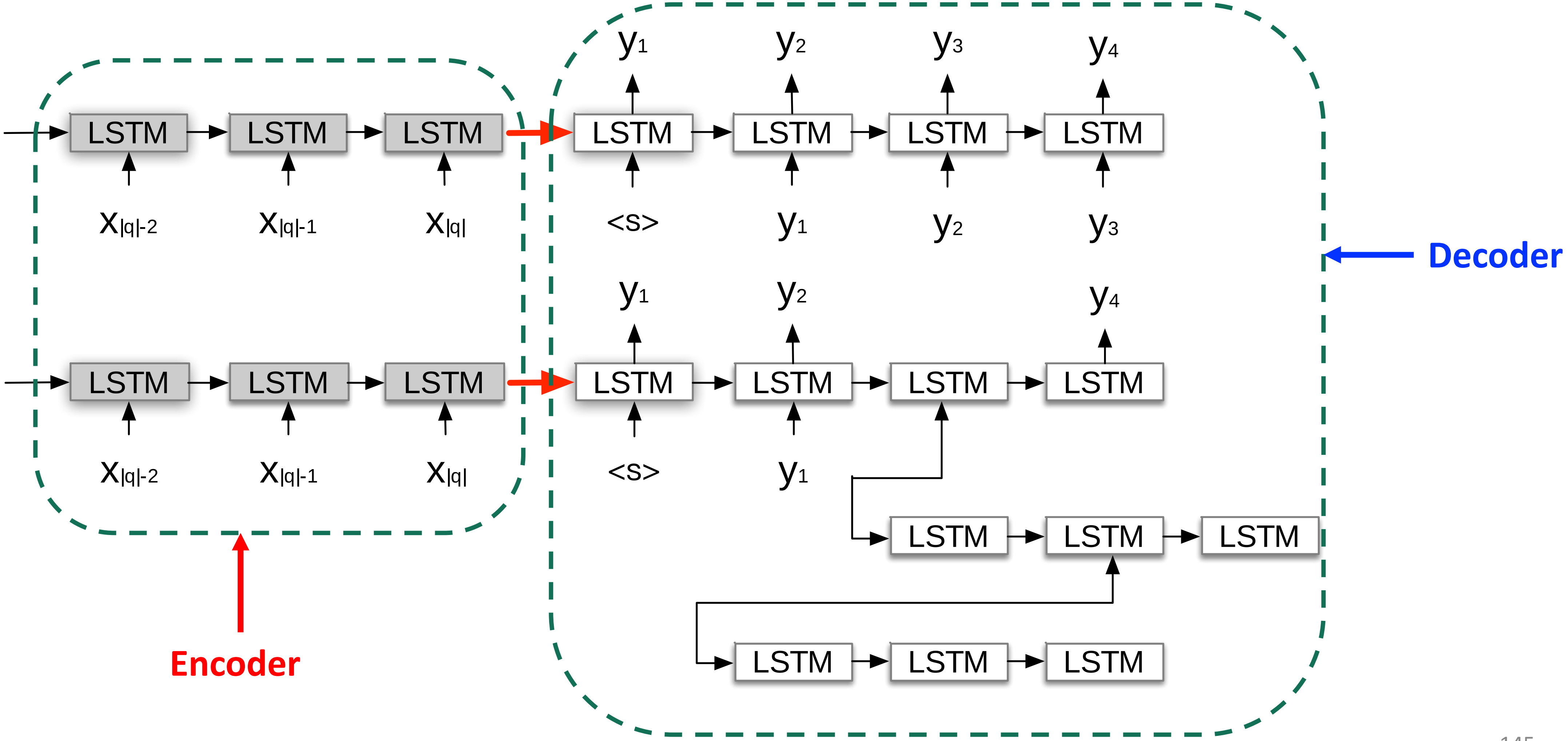
- **Translation** problem
  a) Sequence-to-Sequence model



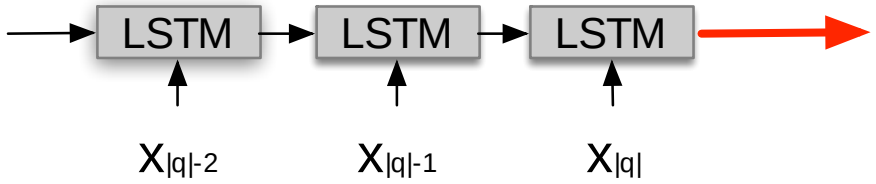
  b) Sequence-to-Tree model



Encoder-Decoder Architecture

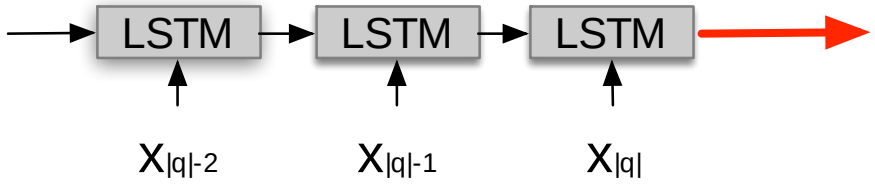


Problem

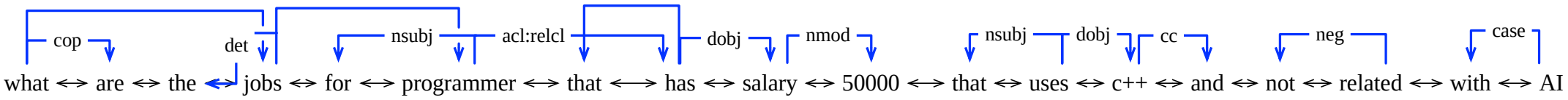


- Sequence LSTM encodes word order features but **loses more complicated syntactic information** such as dependency and constituency trees.
  a) Dependency Feature

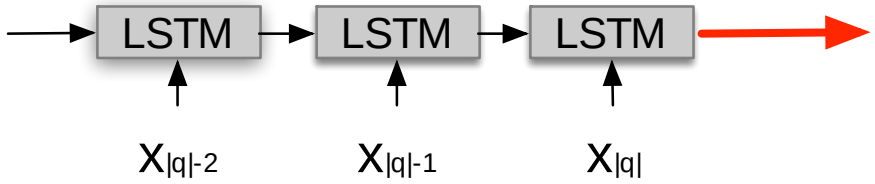
Problem



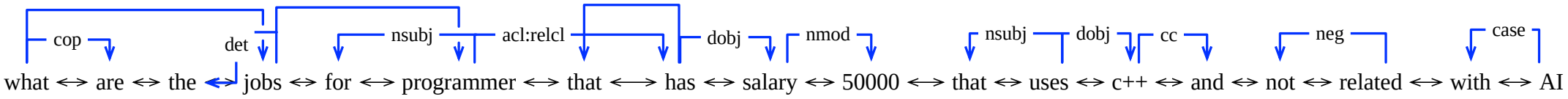
- Sequence LSTM encodes word order features but **loses more complicated syntactic information** such as dependency and constituency trees.
  a) Dependency Feature



Problem

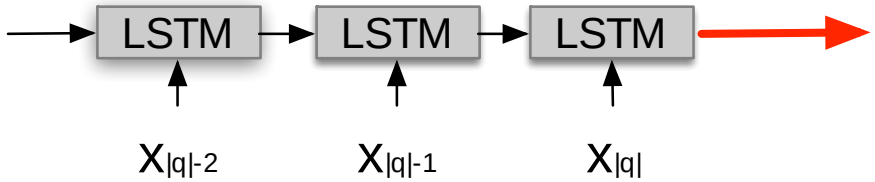


- Sequence LSTM encodes word order features but **loses more complicated syntactic information** such as dependency and constituency trees.
  a) Dependency Feature



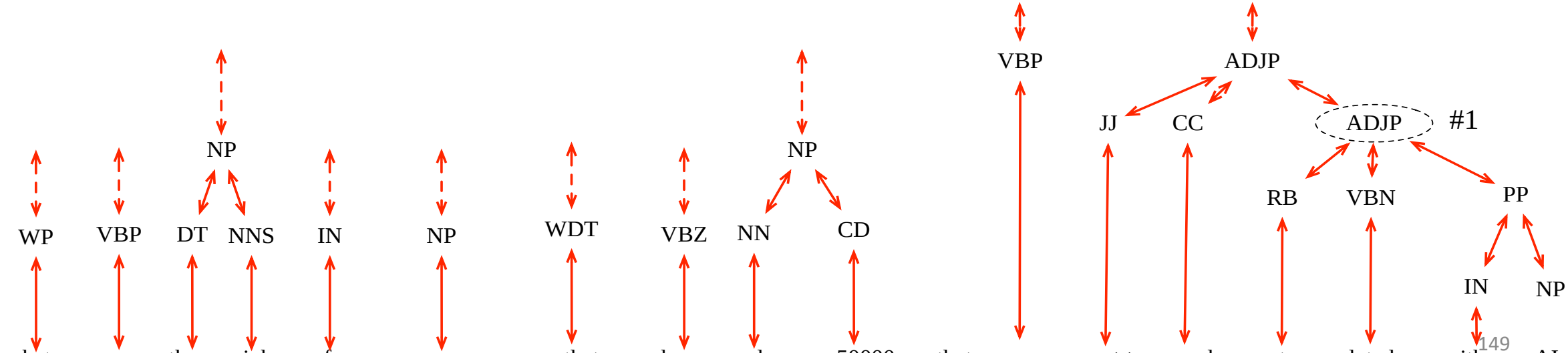
Describes the grammatical relations that hold among a pair of words.

Problem

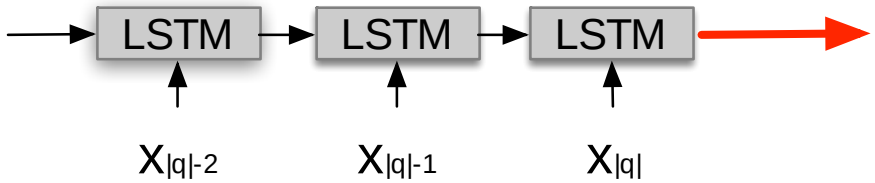


- Sequence LSTM encodes word order features but **loses more complicated syntactic information** such as dependency and constituency trees.

(b) Constituency Feature

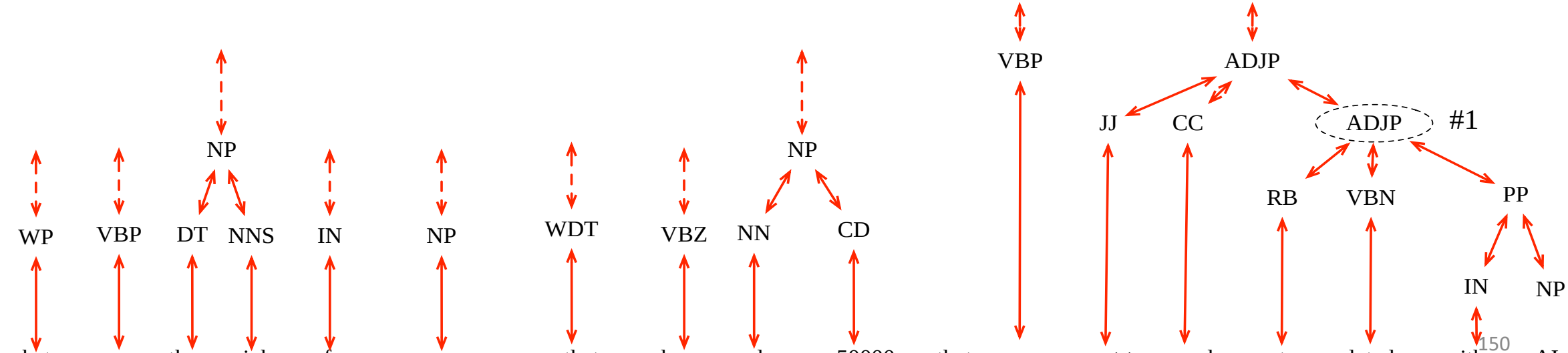


Problem



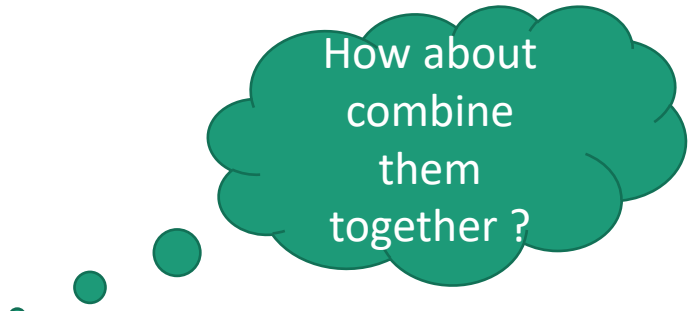
- Sequence LSTM encodes word order features but **loses more complicated syntactic information** such as dependency and constituency trees.

  (b) Constituency Feature represents the sentence structure



Our Approach

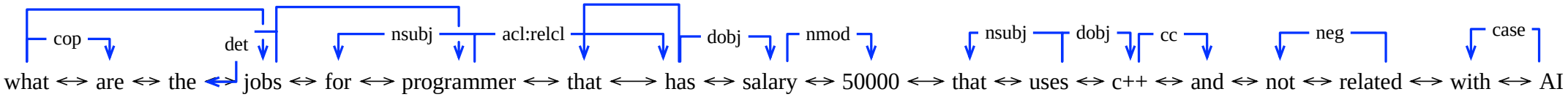
- Syntactic graph
  a) word order
  b) dependency tree
  c) constituency tree



Our Approach

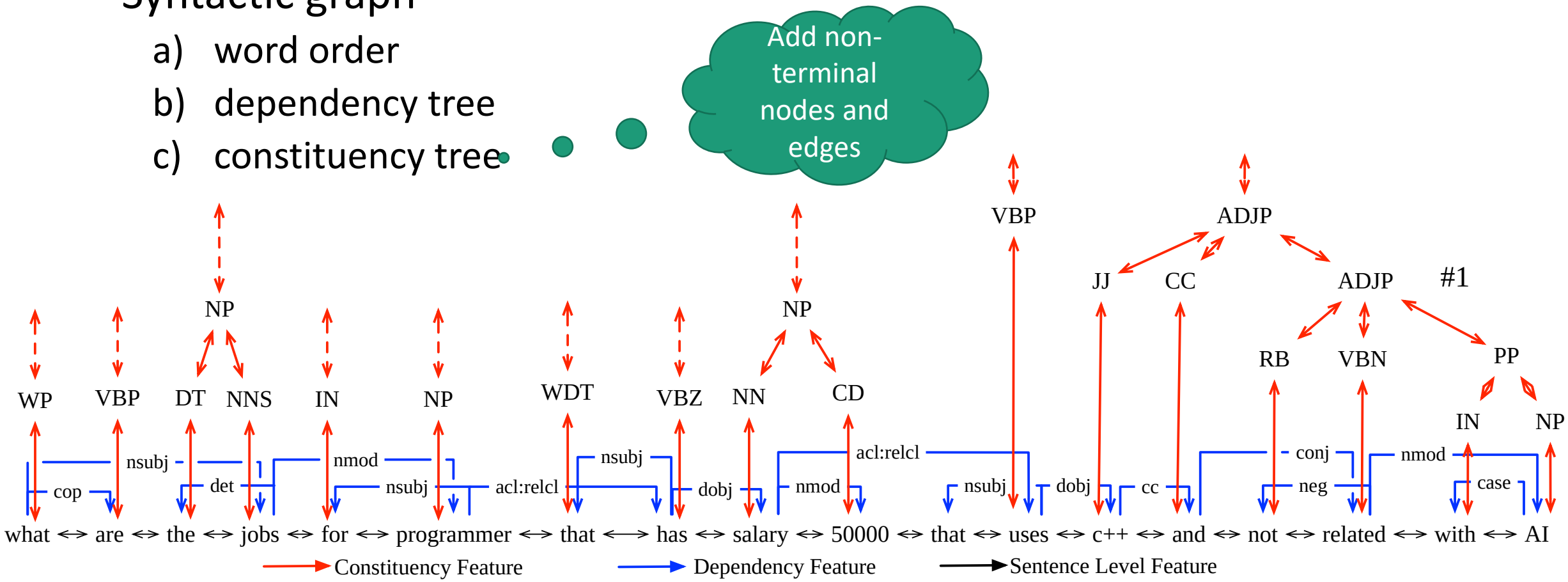
- Syntactic graph
  a) word order
  b) dependency tree
  c) constituency tree

Adding labeled edges between words



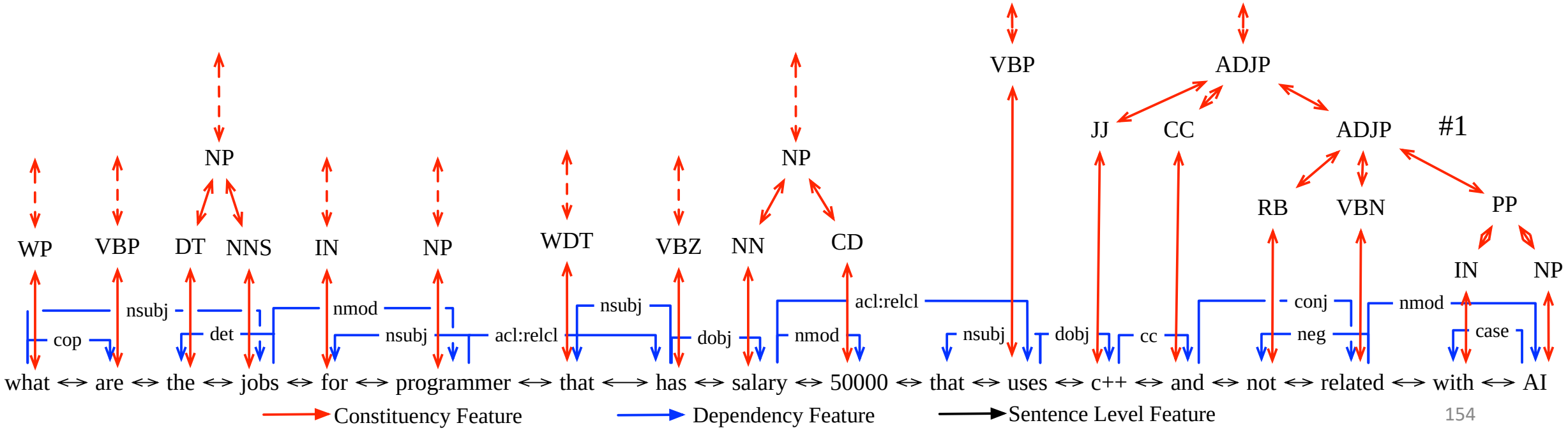
Our Approach

- Syntactic graph
  a) word order
  b) dependency tree
  c) constituency tree

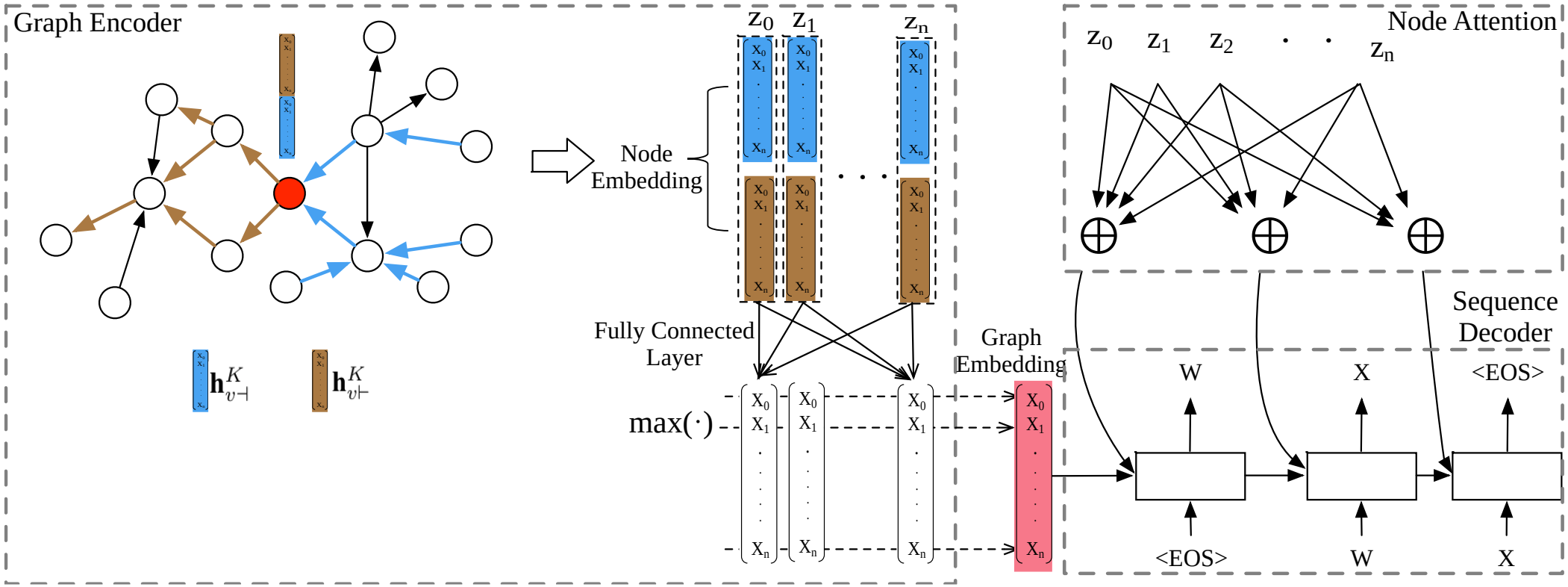


Syntactic Graph Representation

How to model this hybrid graph structure ?



Graph-to-Sequence Model [1]

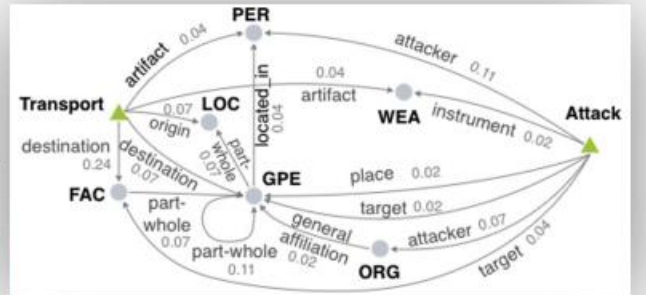


[1] Kun Xu*, Lingfei Wu*, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin (both authors contributed equally), "Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks", arXiv preprint 2018.

Seq2Seq

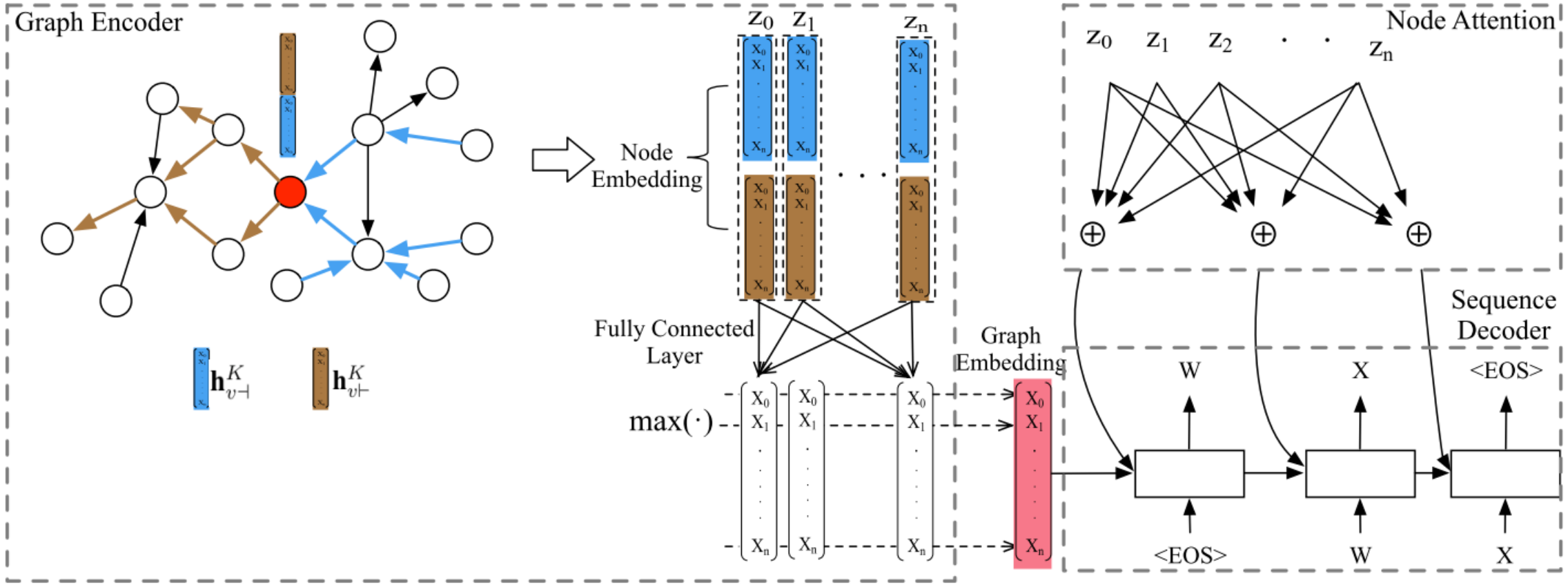
Seq2Seq: Applications and Challenges

- Applications
  - Machine translation
  - Natural language generation
  - Logic form translation
  - Information extraction



- Challenges
  - Only applied to problems whose inputs are represented as sequences
  - Cannot handle more complex structure such as graphs
  - Converting graph inputs into sequences inputs lose information
  - Augmenting original sequence inputs with additional structural information enhances word sequence feature

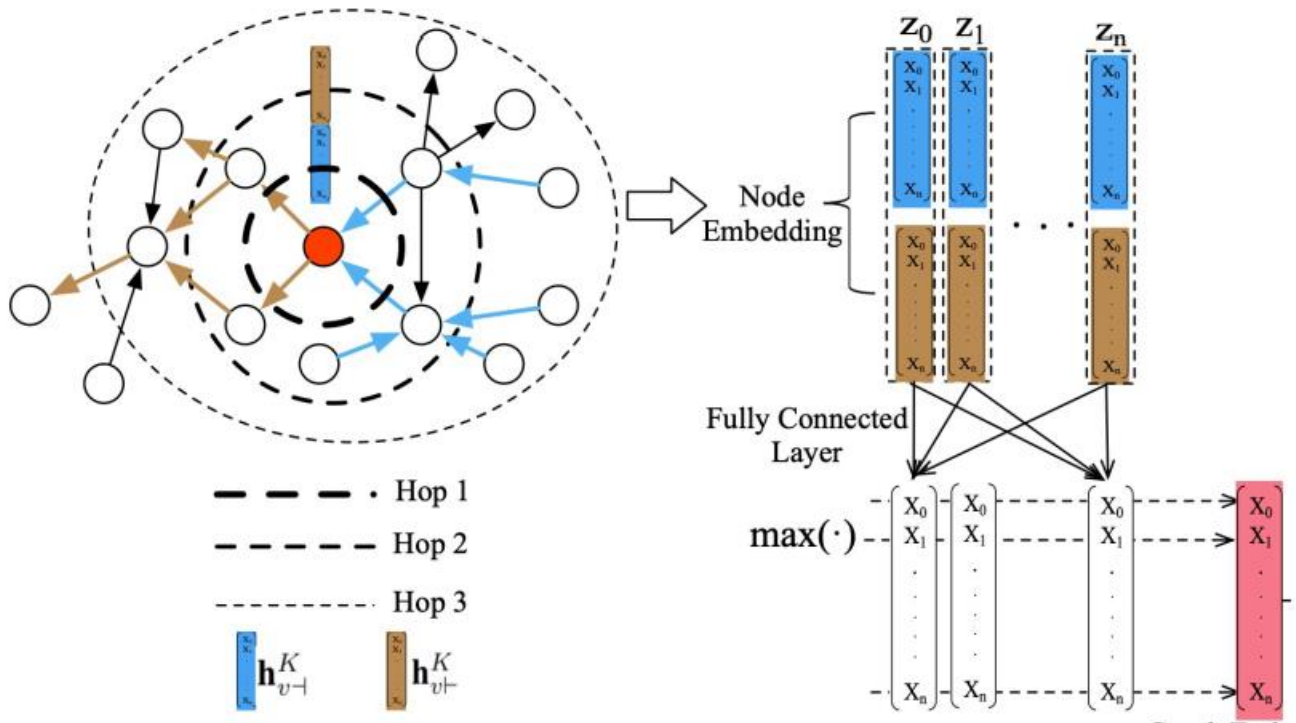
Graph-to-Sequence Model



[1] Kun Xu*, Lingfei Wu*, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin (Equally Contributed), "Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks", arXiv 2018.
[2] Yu Chen, Lingfei Wu** and Mohammed J. Zaki (**Corresponding Author), "Reinforcement Learning Based Graph-to-Sequence Model for Natural Question Generation", ICLR'20.

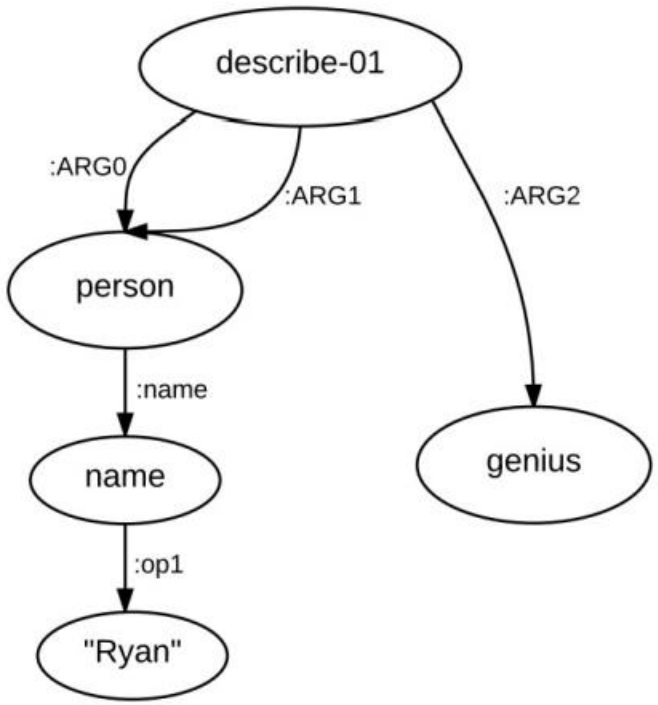
Graph Encoding

- Graph embedding
  - Pooling based graph embedding (*max, min and average pooling*)
  - Node based graph embedding
    - Add one super node which is connected to all other nodes in the graph
    - The embedding of this super node is treated as graph embedding



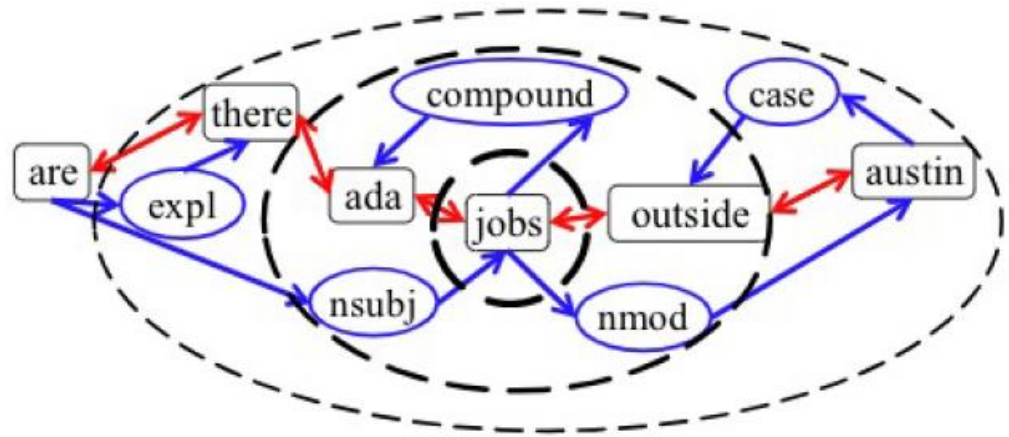
When Shall We Use Graph2Seq?

- Case I: the inputs are naturally or best represented in graph



"Ryan's description of himself: a genius."

- Case II: Hybrid Graph with sequence and its hidden structural information



Augmenting "are there ada jobs outside Austin" with its dependency parsing tree results

Learning Structured Input-Output Translation

- To bridge the semantic gap between the human-readable words and machine-understandable logics.
- Semantic parsing is important for question answering, text understanding
- Automatically solving of MWP is a growing interest.

SP	**Text Input:** what jobs are there for web developer who know 'c++' ? **Structured output:** answer(A , (job (A) , title (A , W) , const (W , 'Web Developer') , language (A , C) , const (C , 'c++')))
MWP	**Text input:** 0.5 of the cows are grazing grass . 0.25 of the cows are sleeping and 9 cows are drinking water from the pond . find the total number of cows . **Structured output:** ((0.5 * x) + (0.25 * x)) + 9.0 = x

Graph and Tree Constructions

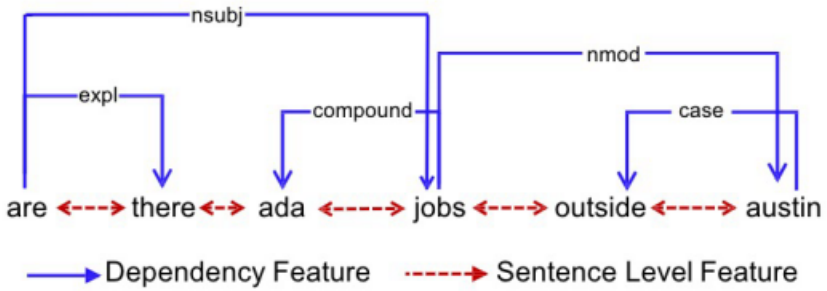


Figure 1: Dependency tree augmented text graph

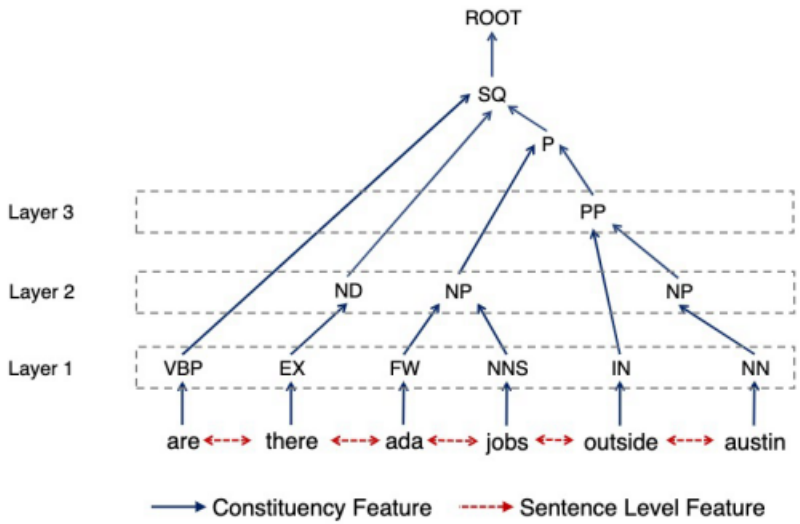


Figure 2: Constituency tree augmented text graph

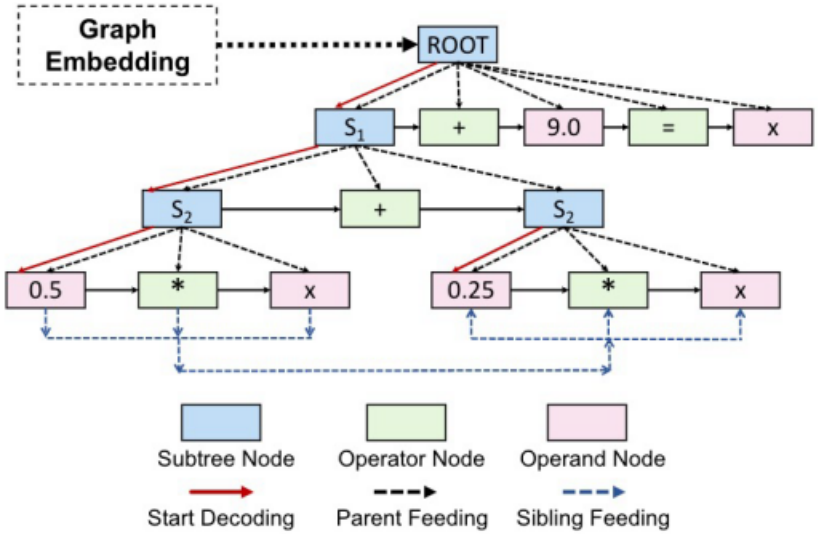
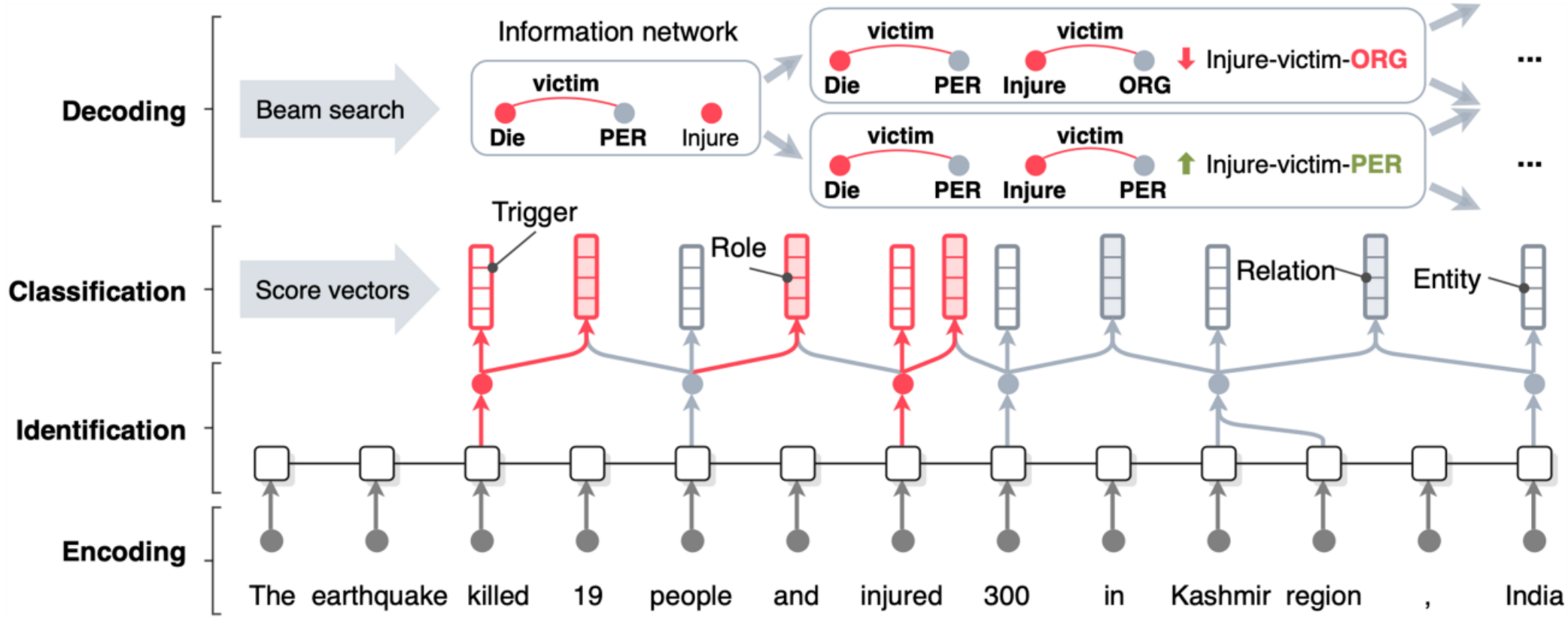


Figure 3: A sample tree output in our decoding process from expression "((**0.5** * **x**) + (**0.25** * **x**)) + **9.0** = **x**"

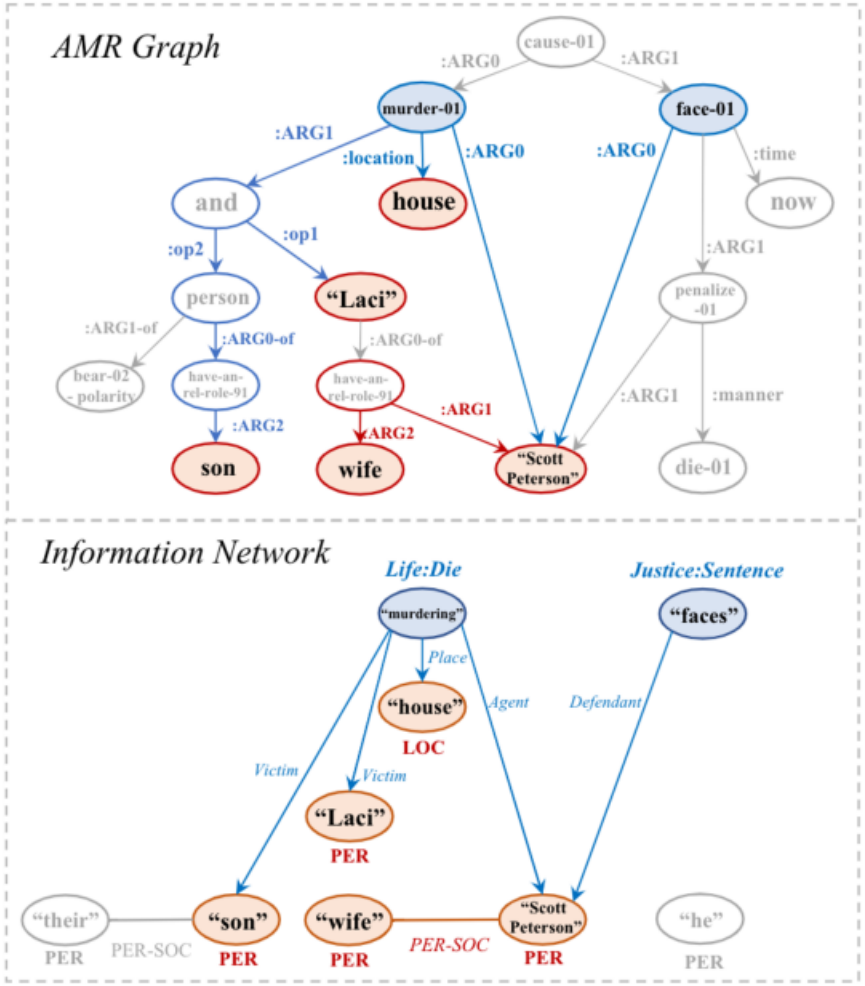
Information Extraction: a Sequence-to-Graph Task



- OneIE [Lin et al., ACL2020] framework extracts the information graph from a given sentence in four steps: encoding, identification, classification, and decoding

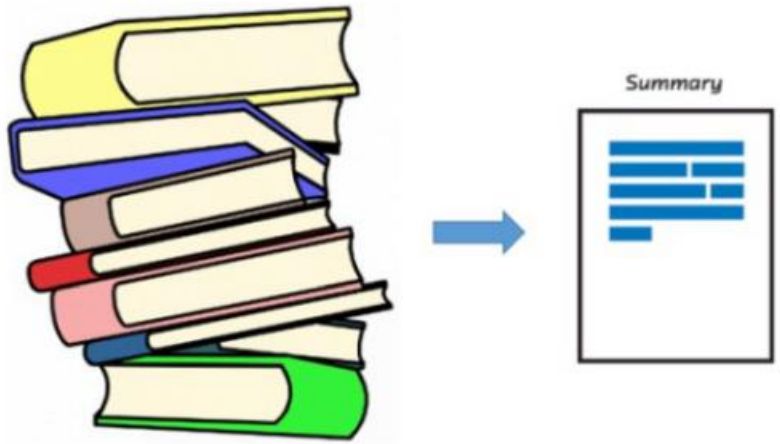
Moving from Seq-to-Graph to Graph-to-Graph

- [Zhang and Ji, NAACL2021]
- Abstract Meaning Representation (AMR):
  - A kind of rich semantic parsing
  - Converts input sentence into a directed and acyclic graph structure with fine-grained node and edge type labels
- AMR parsing shares inherent similarities with information network (IE output)
  - Similar node and edge semantics
  - Similar graph topology
- Semantic graphs can better capture non-local context in a sentence
- Exploit the similarity between AMR and IE to help on joint information extraction



Summarization

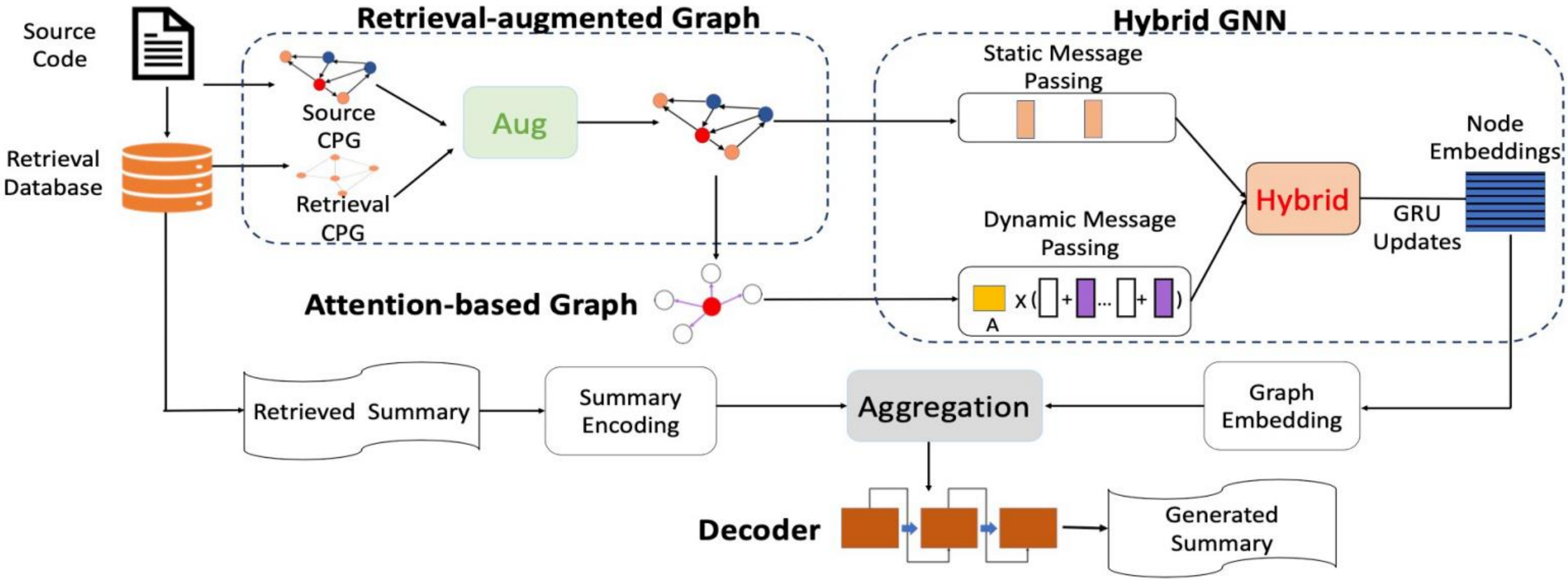
Summarization



- Input
  - A document, dialogue, code or multiple ones
- Output
  - A succinct sentence or paragraph

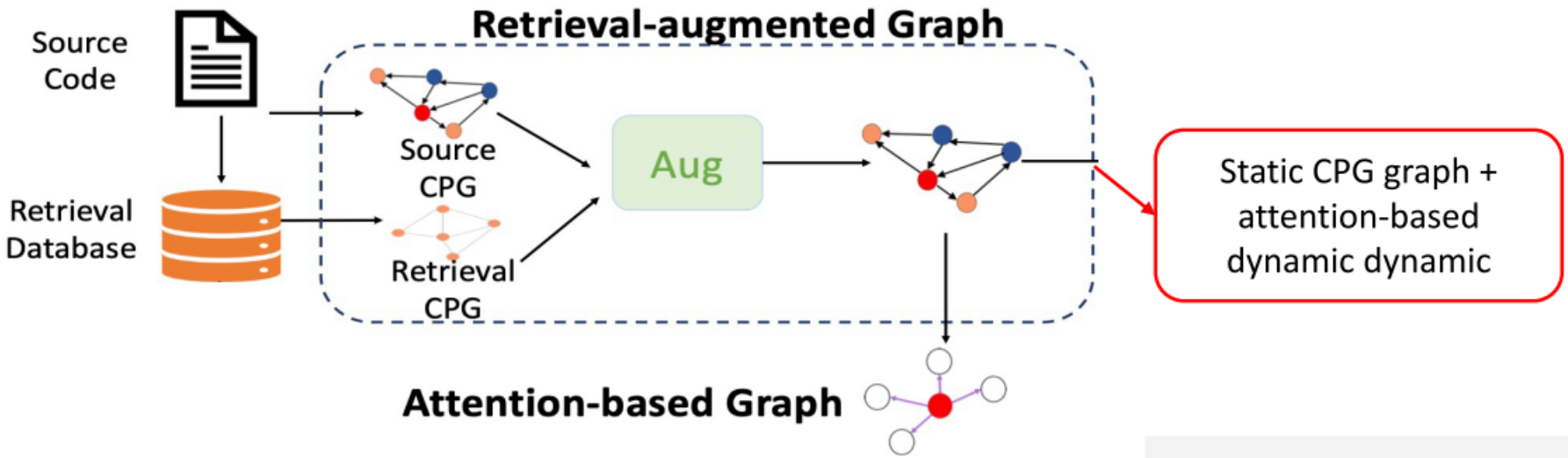
Ref: https://www.queppelin.com/how-nlp-is-helping-in-automatic-text-summarization-2, https://www.fiverr.com/yosree22/summarize-any-long-text-or-article-for-you

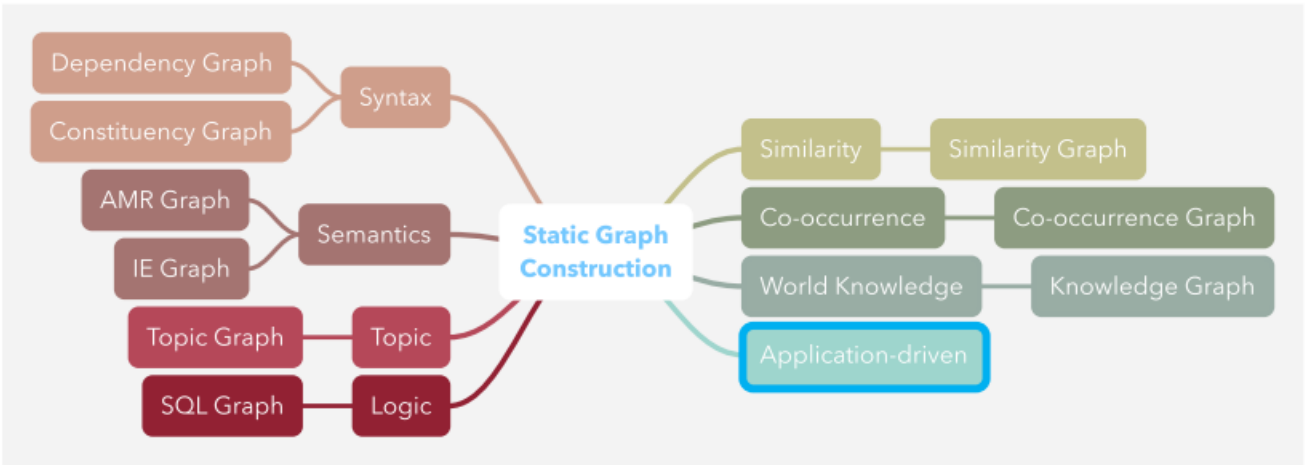
GNN for Code Summarization [Liu et al. ICLR'21]

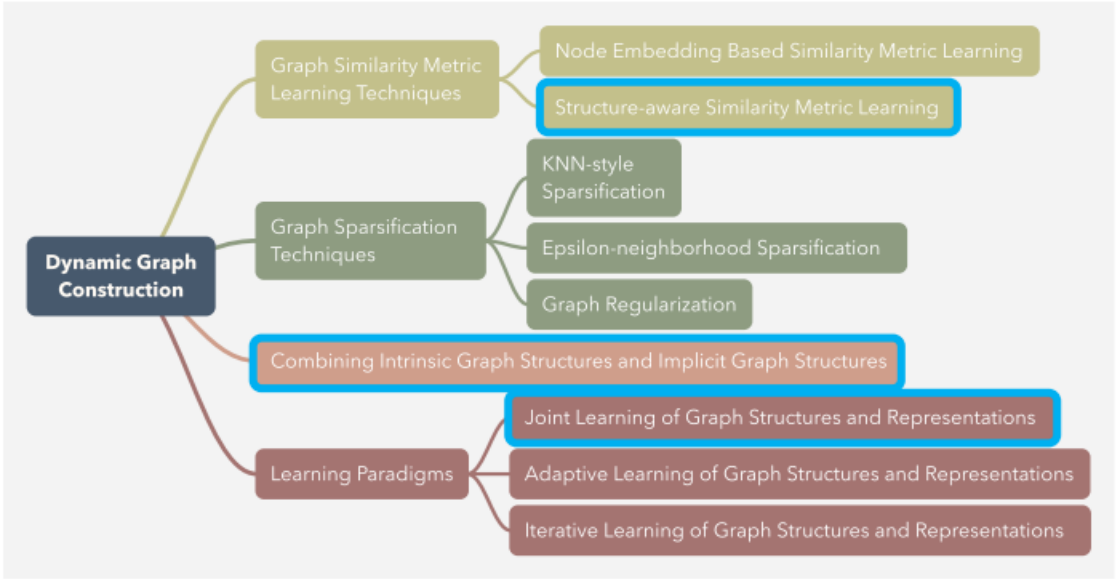


Liu et al. "Retrieval-Augmented Generation for Code Summarization via Hybrid GNN". ICLR 2021.

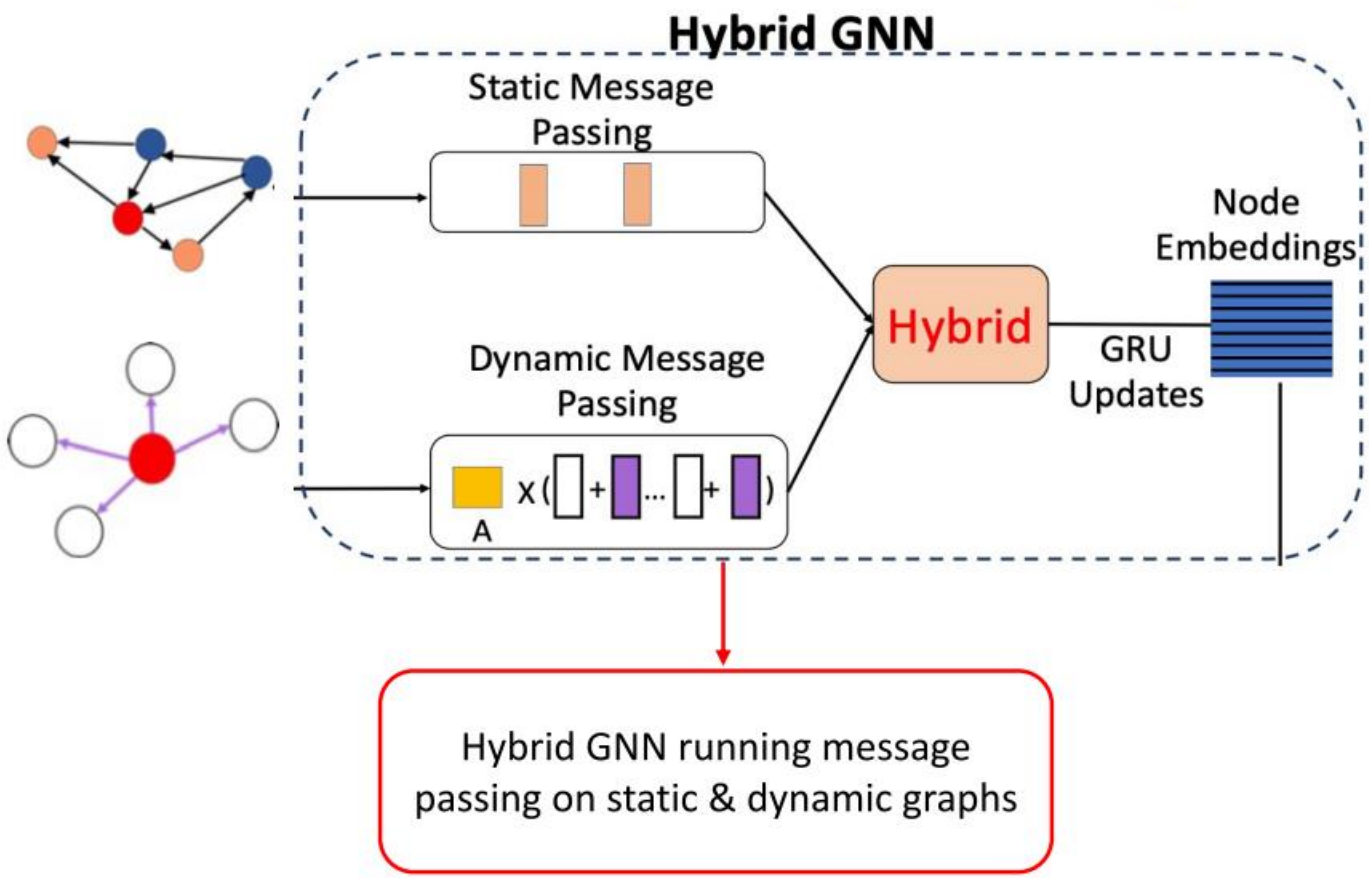
GNN for Code Summarization [Liu et al. ICLR'21]

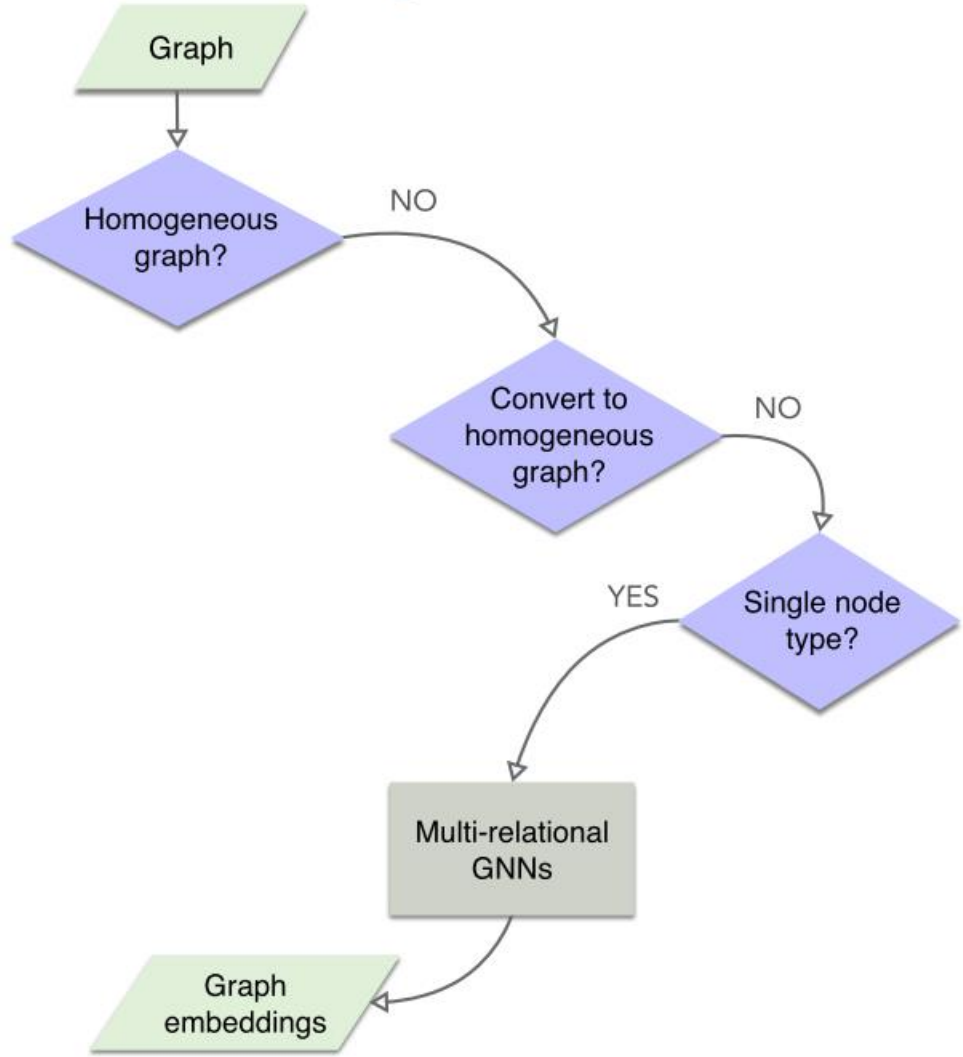






GNN for Code Summarization [Liu et al. ICLR'21]

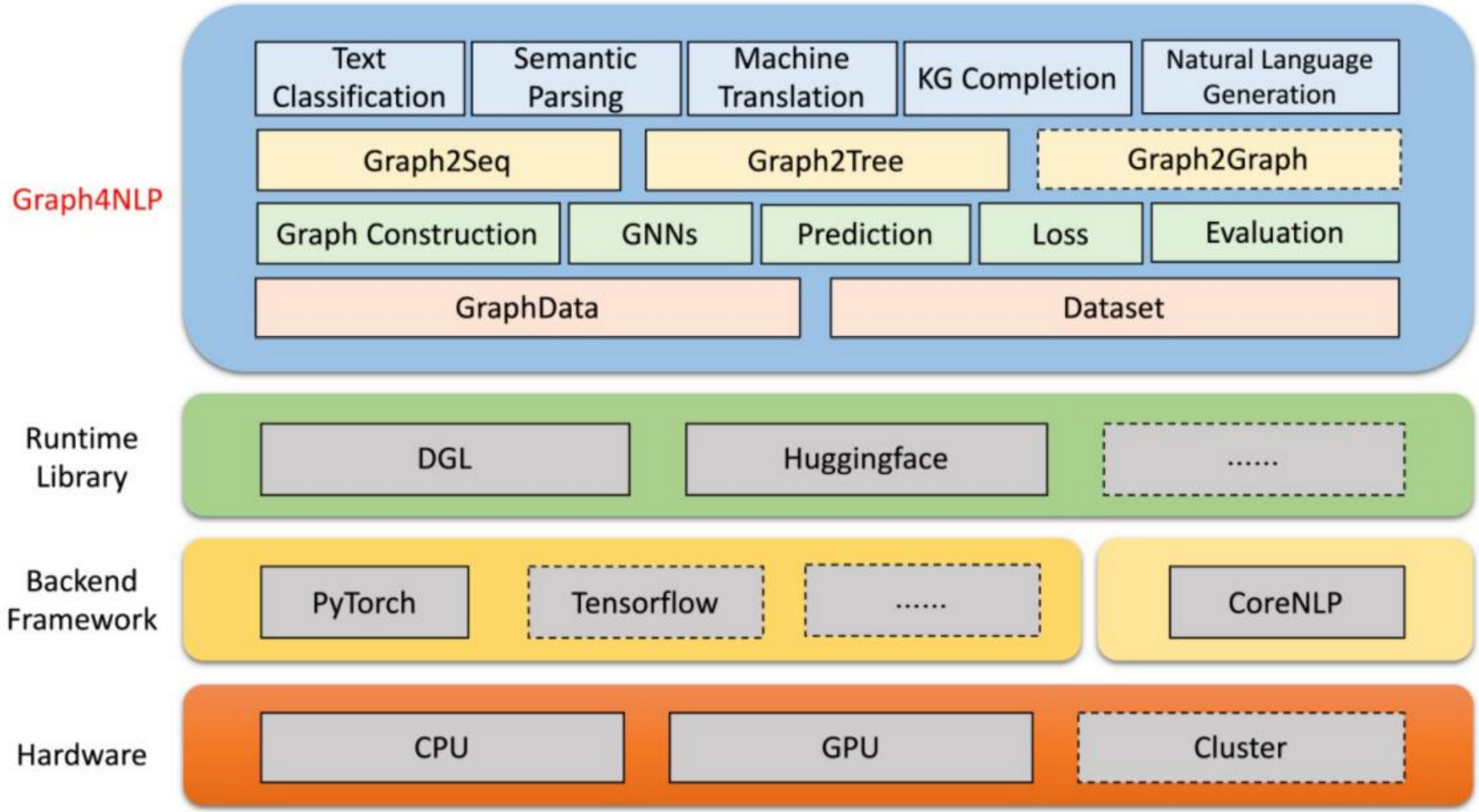




Graph4NLP

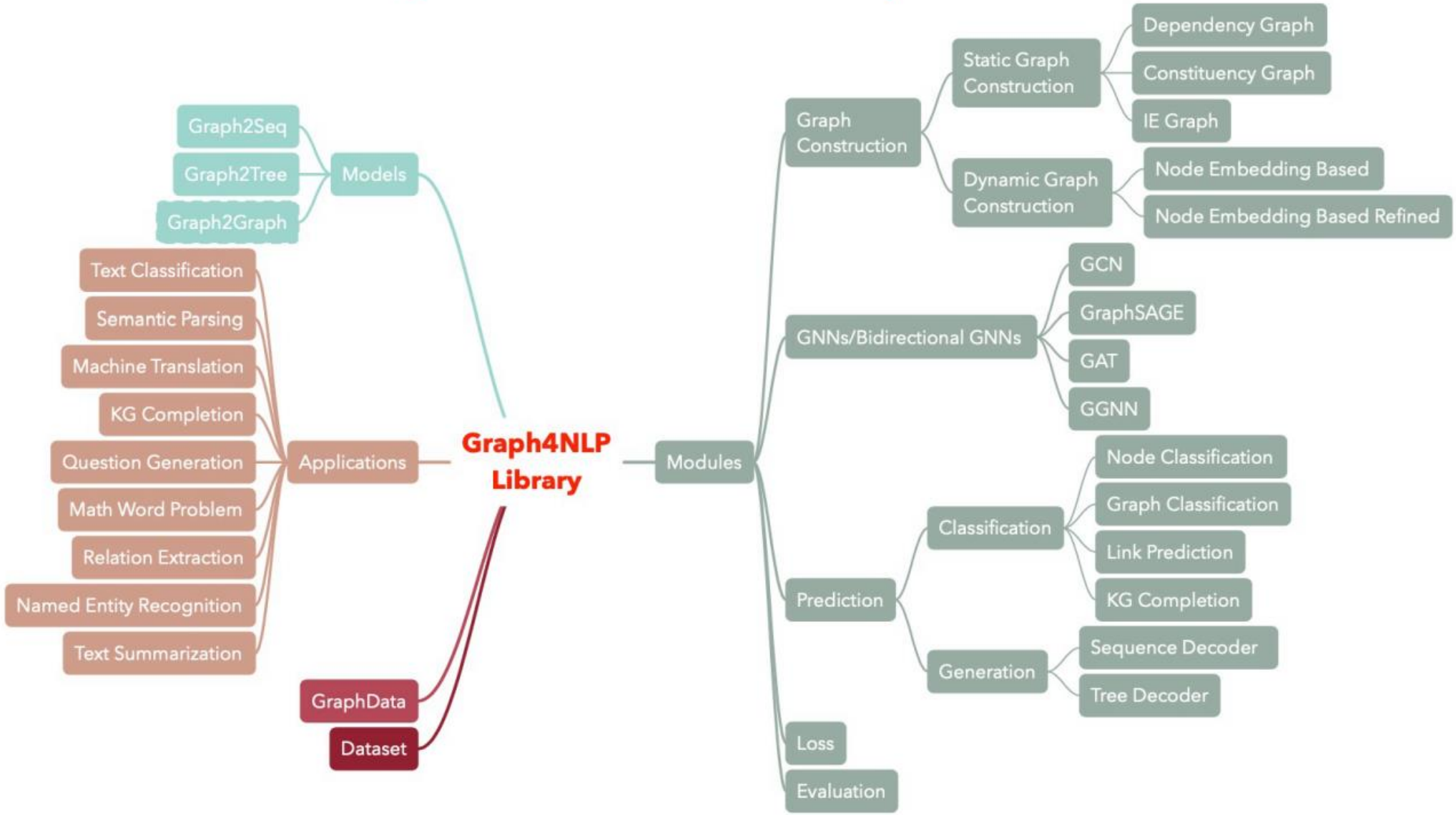
Graph4NLP: A Library for Deep Learning on Graphs for NLP

Overall Architecture of Graph4NLP Library



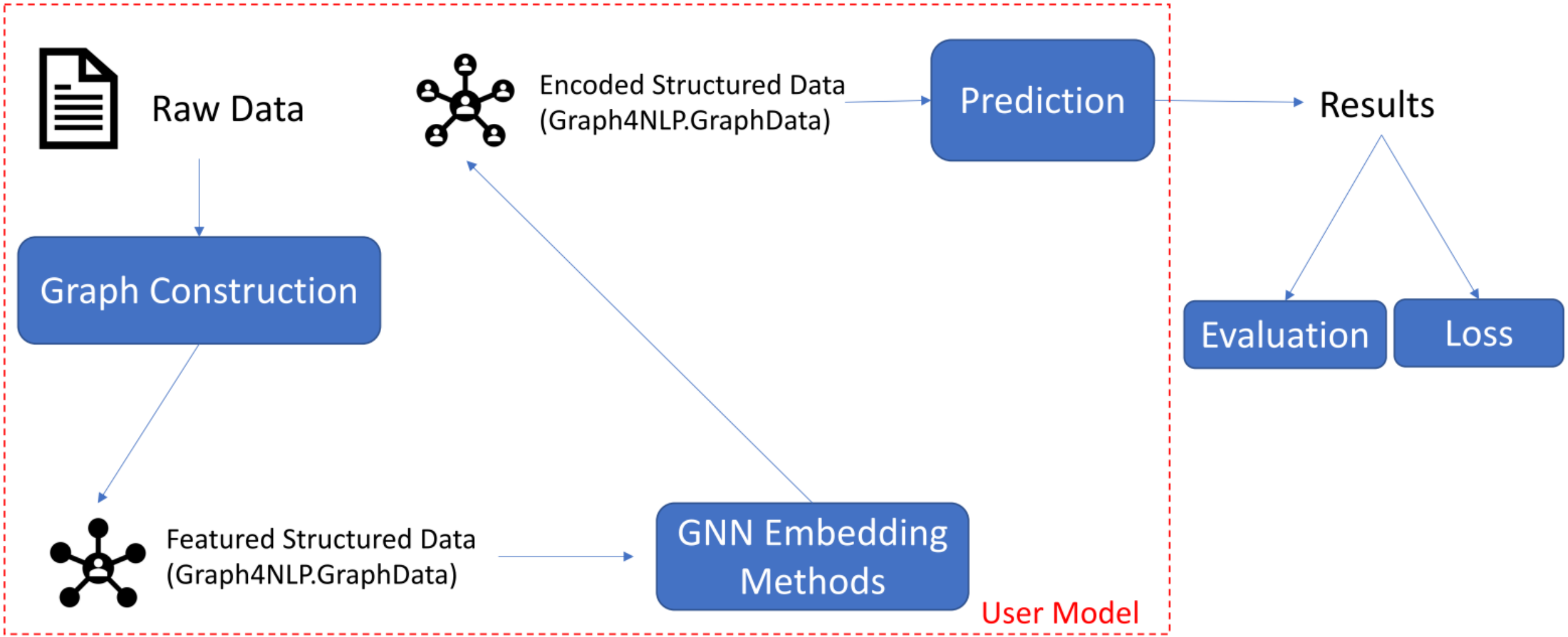
DGL: https://github.com/dmlc/dgl, DIG: https://github.com/divelab/DIG, Huggingface: https://github.com/huggingface/transformers

Dive Into Graph4NLP Library



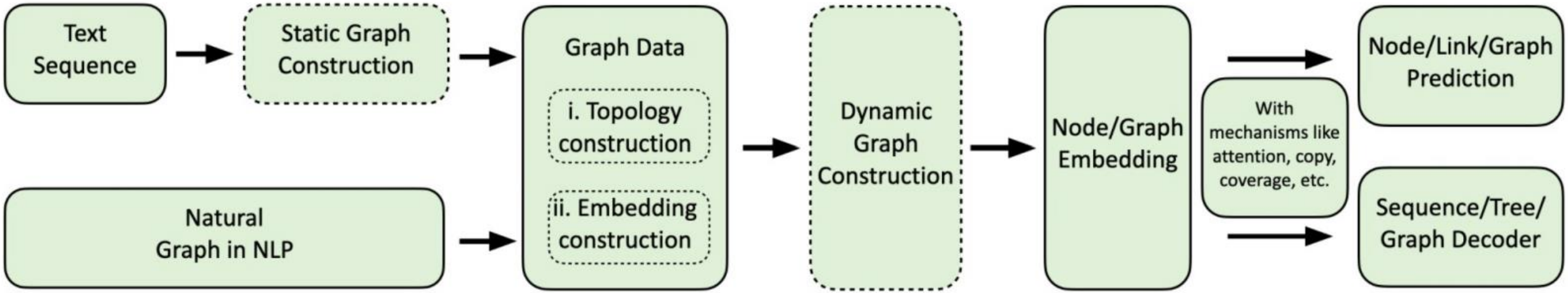
Graph4NLP

Data Flow of Graph4NLP



Graph4NLP

Computing Flow of Graph4NLP



Graph4NLP

Performance of Built-in NLP Tasks

Task	Dataset	GNN Model	Graph construction	Evaluation	Performance
Text classification	TRECT CAirline CNSST	GAT	Dependency	Accuracy	0.948 0.769 0.538
Semantic Parsing	JOBS	SAGE	Constituency	Execution accuracy	0.936
Question generation	SQuAD	GGNN	Dependency	BLEU-4	0.15175
Machine translation	IWSLT14	GCN	Dynamic	BLEU-4	0.3212
Summarization	CNN(30k)	GCN	Dependency	ROUGE-1	26.4
Knowledge graph completion	Kinship	GCN	Dependency	MRR	82.4
Math word problem	MAWPS MATHQA	SAGE	Dynamic	Solution accuracy Exact match	76.4 61.07

Demo 1: Building a Text Classification Application

1) git clone https://github.com/graph4ai/graph4nlp_demo
2) follow Get Started instructions in README

Demo 1: Building a Text Classification Application

```
def forward(self, graph_list, tgt=None, require_loss=True):
    # build graph topology
    batch_gd = self.graph_topology(graph_list)

    # run GNN encoder
    self.gnn(batch_gd)

    # run graph classifier
    self.clf(batch_gd)
    logits = batch_gd.graph_attributes['logits']

    if require_loss:
        loss = self.loss(logits, tgt)
        return logits, loss
    else:
        return logits
```

Model arch

https://github.com/graph4ai/graph4nlp_demo/tree/main/NAACL2021_demo

Demo 1: Building a Text Classification Application

Graph construction API, various built-in options, can be customized

```
self.graph_topology = DependencyBasedGraphConstruction(
                        embedding_style=embedding_style,
                        vocab=vocab.in_word_vocab,
                        hidden_size=config['num_hidden'],
                        word_dropout=config['word_dropout'],
                        rnn_dropout=config['rnn_dropout'],
                        fix_word_emb=not config['no_fix_word_emb'],
                        fix_bert_emb=not config.get('no_fix_bert_emb', False))
```

https://github.com/graph4ai/graph4nlp_demo/tree/main/NAACL2021_demo

Demo 1: Building a Text Classification Application

GNN API, various built-in options, can be customized

```
self.gnn = GraphSAGE(config['gnn_num_layers'],
             config['num_hidden'],
             config['num_hidden'],
             config['num_hidden'],
             config['graphsage_aggreagte_type'],
             direction_option=config['gnn_direction_option'],
             feat_drop=config['gnn_dropout'],
             bias=True,
             norm=None,
             activation=F.relu,
             use_edge_weight=use_edge_weight)
```

https://github.com/graph4ai/graph4nlp_demo/tree/main/NAACL2021_demo

Demo 1: Building a Text Classification Application

Prediction API, various built-in options, can be customized

```
self.clf = FeedForwardNN(2 * config['num_hidden'] \
                if config['gnn_direction_option'] == 'bi_sep' \
                else config['num_hidden'],
                config['num_classes'],
                [config['num_hidden']],
                graph_pool_type=config['graph_pooling'],
                dim=config['num_hidden'],
                use_linear_proj=config['max_pool_linear_proj'])
```

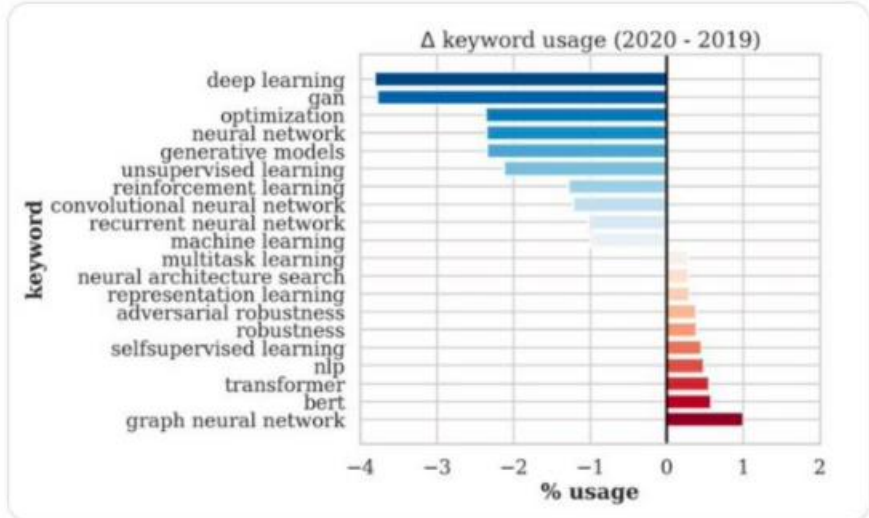
https://github.com/graph4ai/graph4nlp_demo/tree/main/NAACL2021_demo

Future Directions

- The Rise of GNN + NLP

#ICLR2020 submissions on graph neural networks, NLP and robustness have the greatest growth. @iclr_conf @openreviewnet

[Vashishth et al. EMNLP'19 Tutorial]



- Graph Construction for NLP
  - Dynamic graph construction are largely underexplored!
  - How to effectively combine advantages of static graph and dynamic graph?
  - How to construct heterogeneous dynamic graph?
  - How to make dynamic graph construction itself scalable?

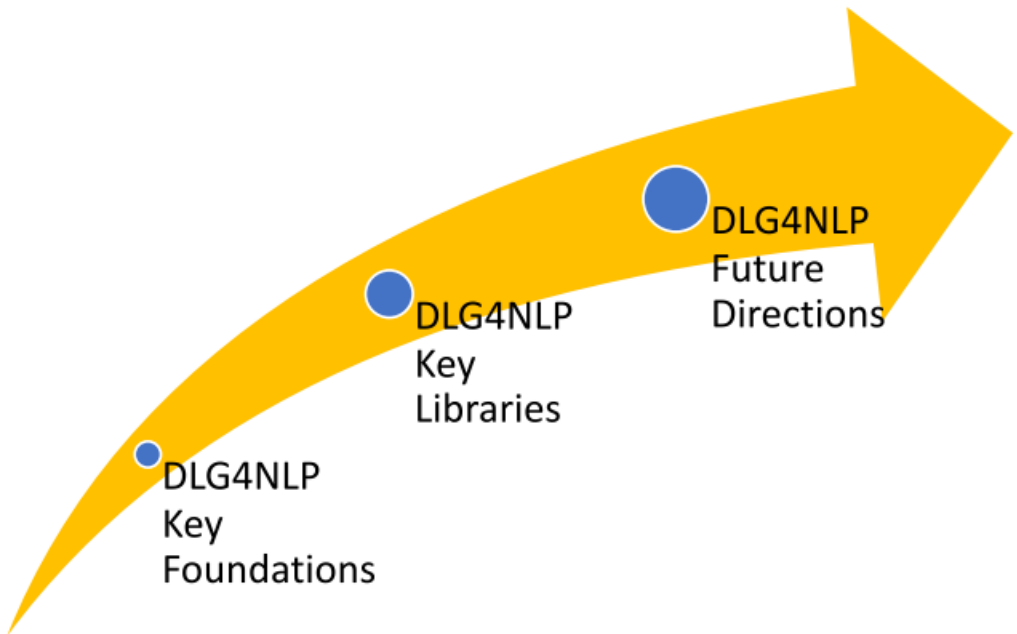
Future Directions

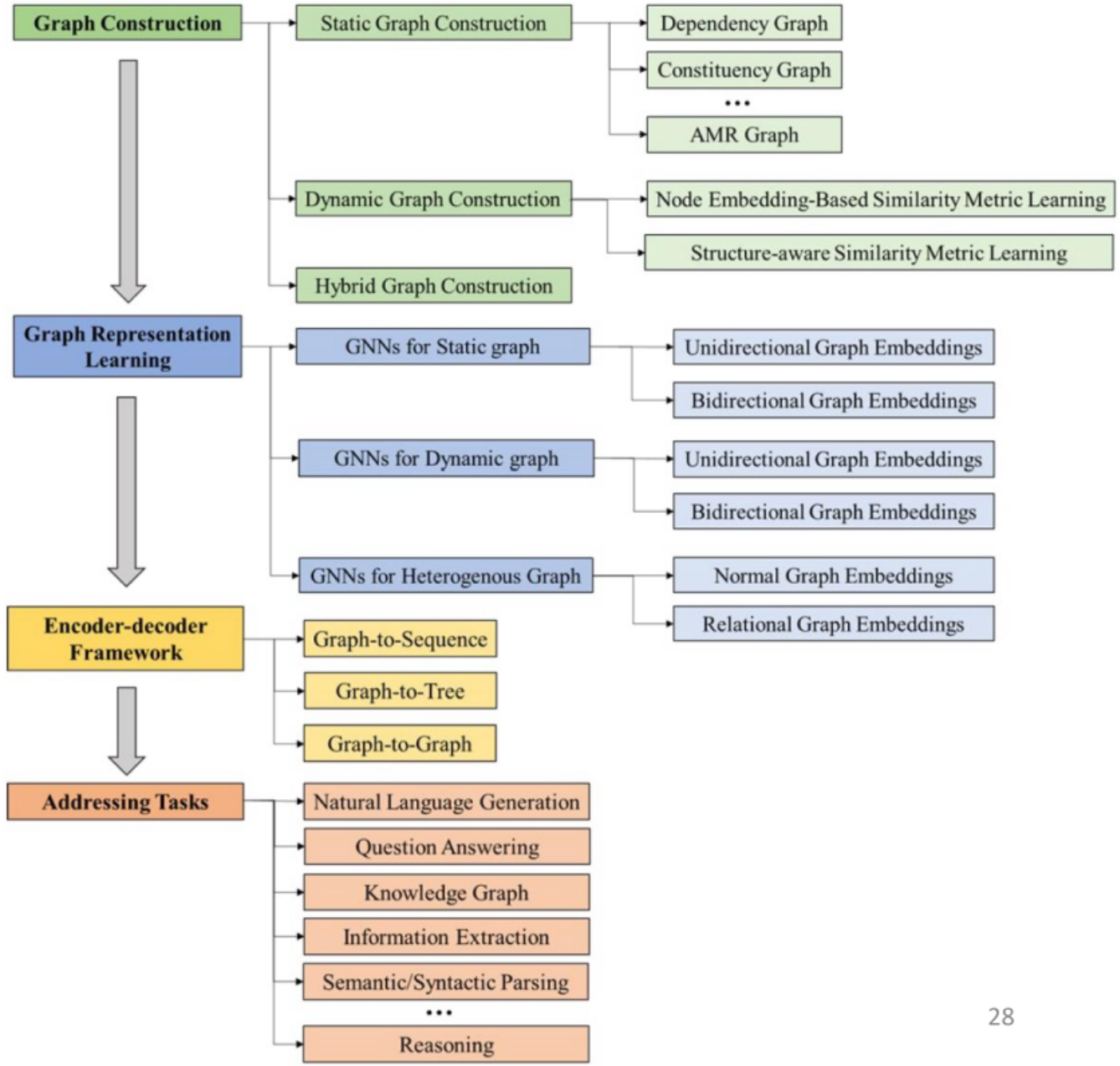
- Scaling GNNs to Large Graphs
  - Most existing multi-relational or heterogeneous GNNs will have scalability issues when applied to large graphs in NLP such as KGs (> 1m)
- GNNs + Transformer in NLP
  - How to effectively combine the advantages of GNNs and Transformer?
  - Is graph transformer the best way to utilize?
- Pretraining GNNs for NLP
  - Information Retrieval/ Search

Future Directions

- Graph-to-graph Learning in NLP
  - How to effectively develop Graph-to-Graph models for solving graph transformation problem in NLP (i.e. information extraction)?
- Joint Text and KG Reasoning in NLP
  - Joint text and KG reasoning is less explored although GNNs for multi-hop reasoning gains popularity
- Incorporate Source and Context into Knowledge Graph Construction and Verification

DLG4NLP: A Roadmap





Summary

- The solution to many applications can be formulated as graphlearning problems.
- Graph neural networks are a new technique for graph learning.
  - They have multiple advantages over traditional methods.
- We demonstrate how GNNs are used in multiple graph tasks and how these GNN models can be trained.

References (Slides primarily borrowed from…)

- Overview of Graph Neural Networks - George Karypis
- Graph Neural Networks: Models and Applications - Yao Ma, et al
- GraphSAGE: Deep Learning for Relational Data - Jure Leskovec
- Deep Learning on Graphs in Natural Language Processing and Computer Vision - Lingfei Wu, IBM Research AI
- Deep Learning on Graphs for Natural Language Processing - Lingfei Wu, Yu Chen, Heng Ji, and Yunyao Li, NAACL-2021 Tutorial