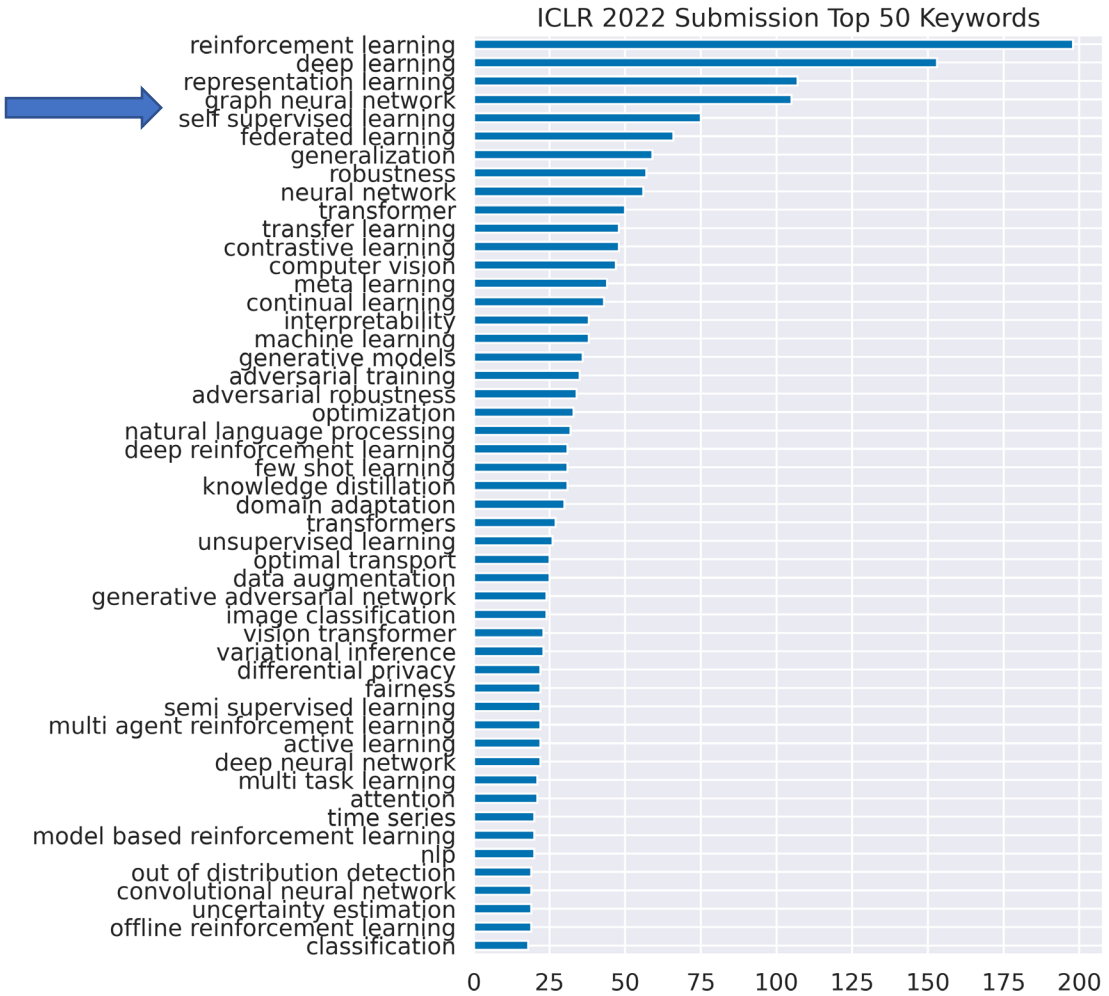
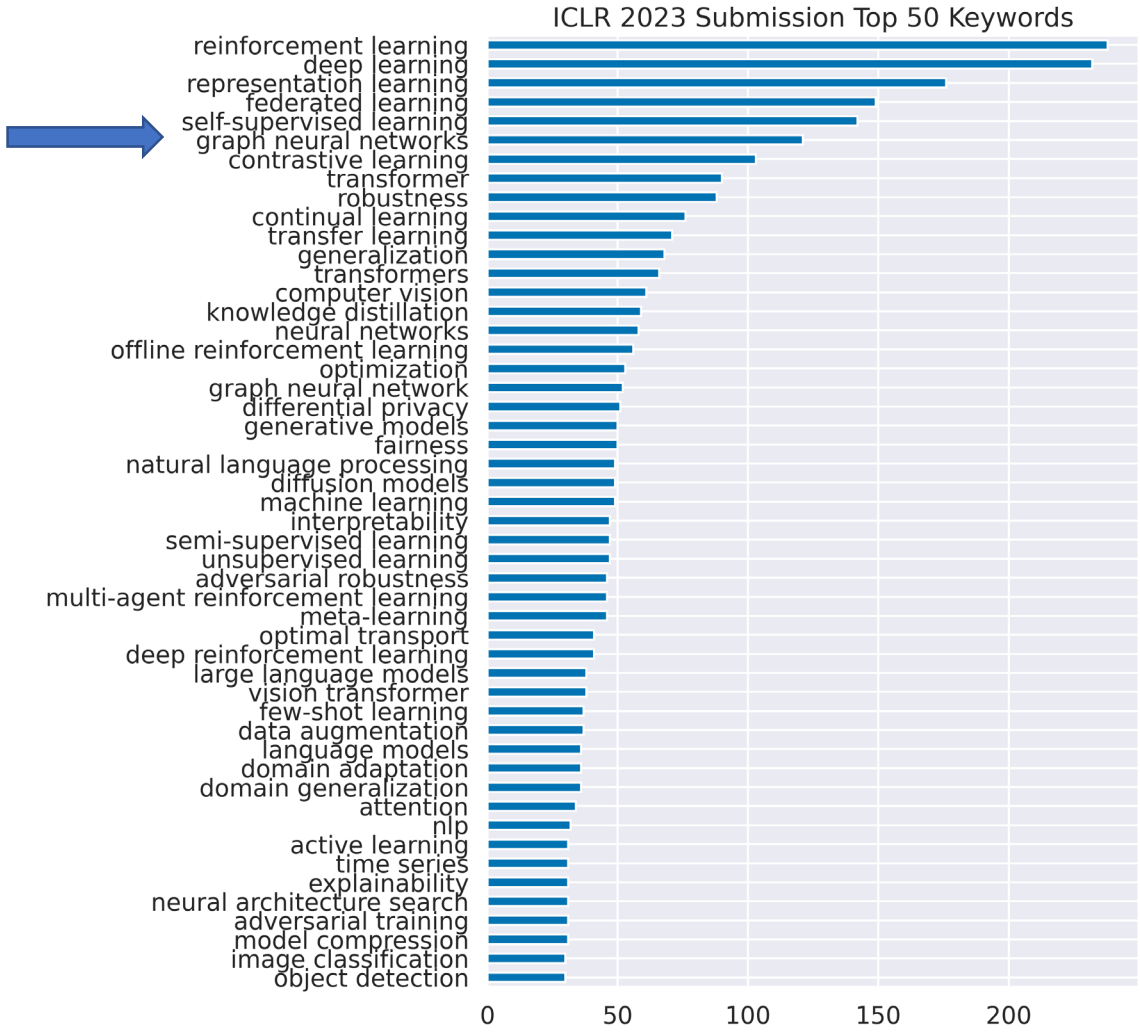
Graph Neural Networks

上海交通大学
约翰·霍普克罗夫特
计算机科学中心
John Hopcroft Center for Computer Science

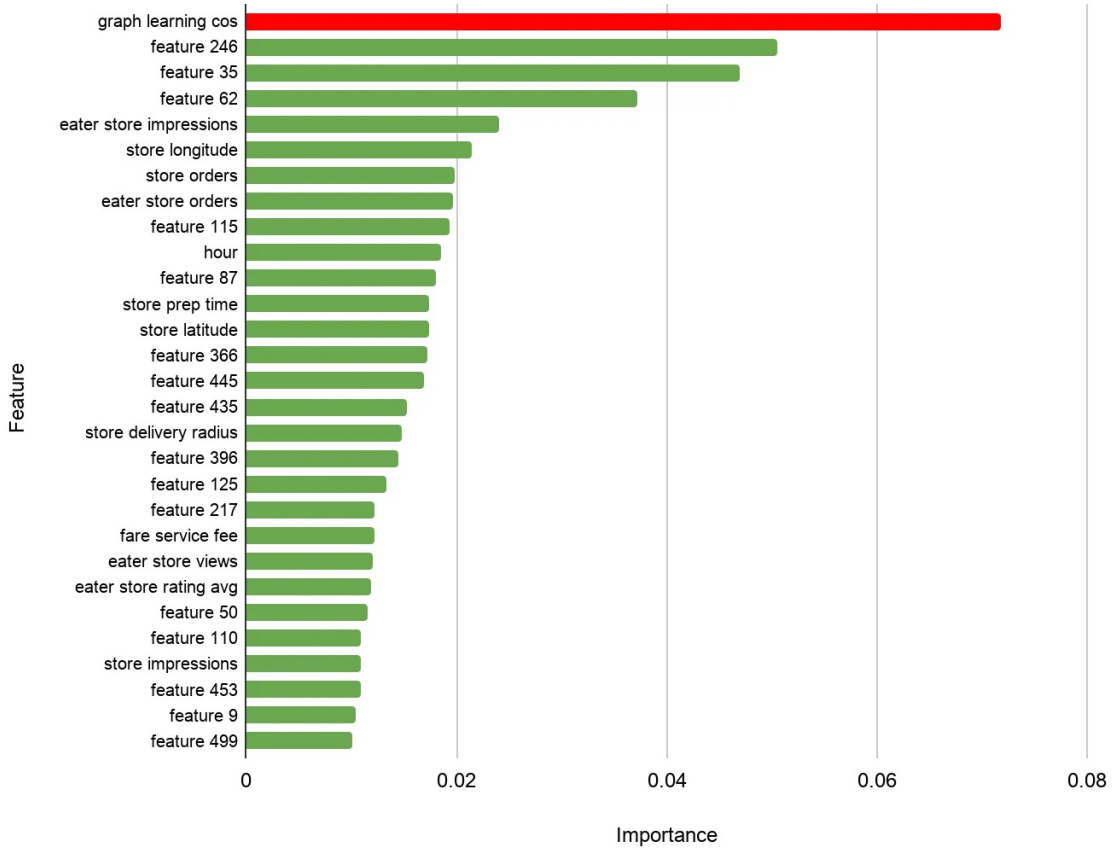
GNN is a hot topic in recent years



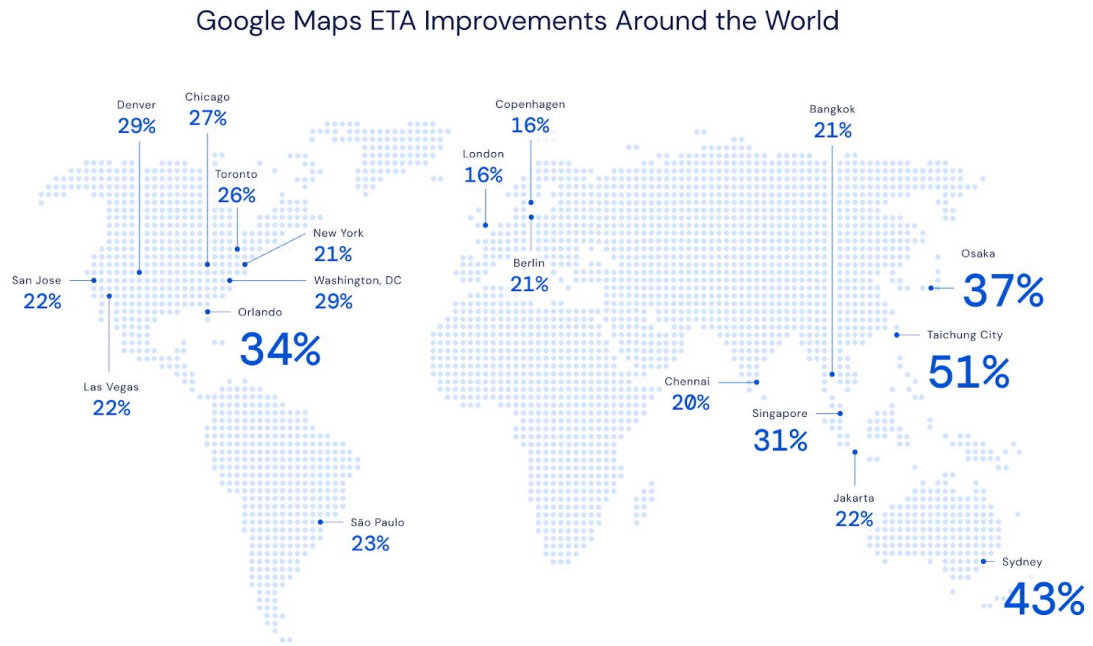


Applications

Uber Eats recommendation system 20% ↑



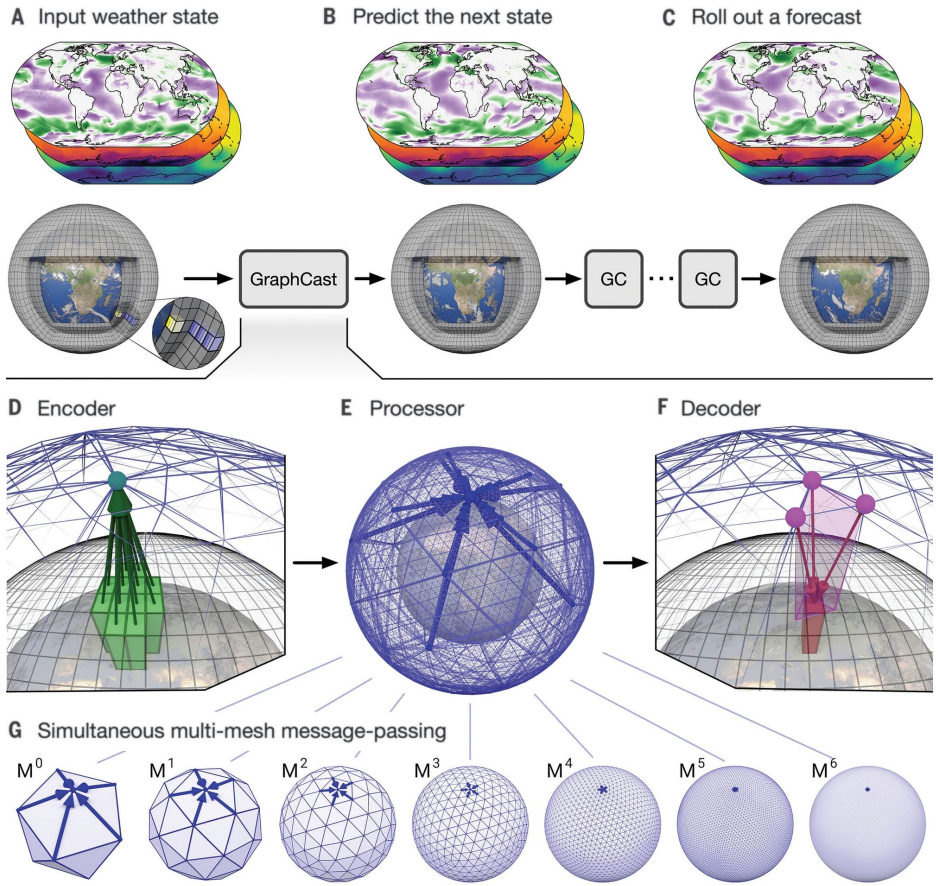
Google Maps ETA accuracy 50% ↑



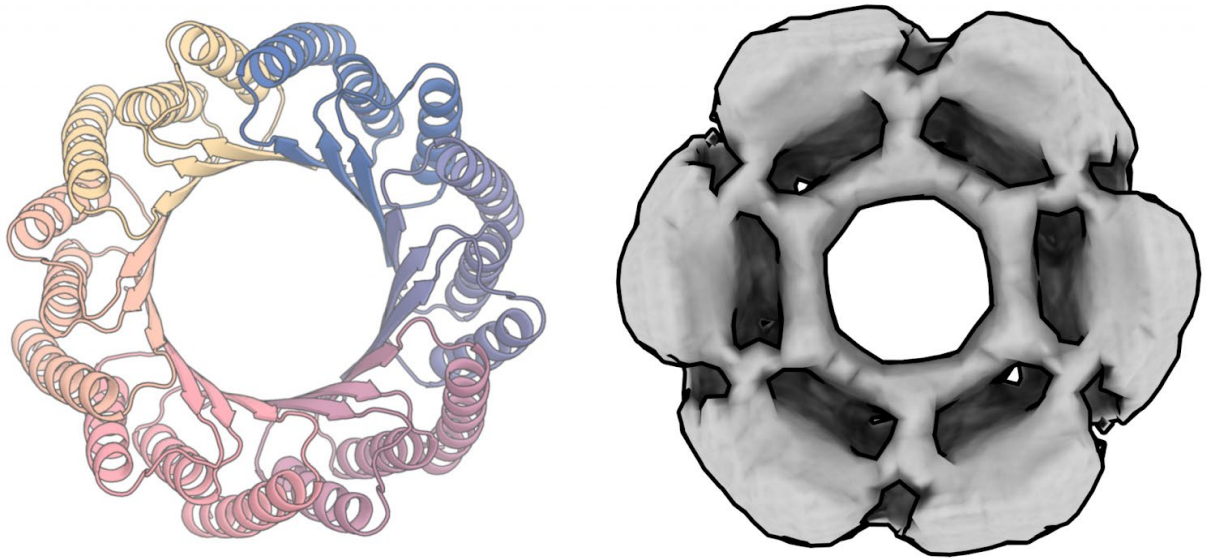
https://www.assemblyai.com/blog/ai-trends-graph-neural-networks/

Applications

DeepMind GraphCast Climate Predication



Baker Lab Protein Design 5x improvement



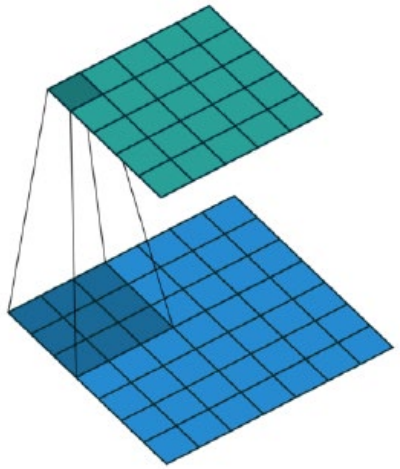
https://www.science.org/doi/10.1126/science.adi2336?ref=assemblyai.com
https://www.bakerlab.org/wp-content/uploads/2022/11/Diffusion_preprint_12012022.pdf?ref=assemblyai.com

Setup

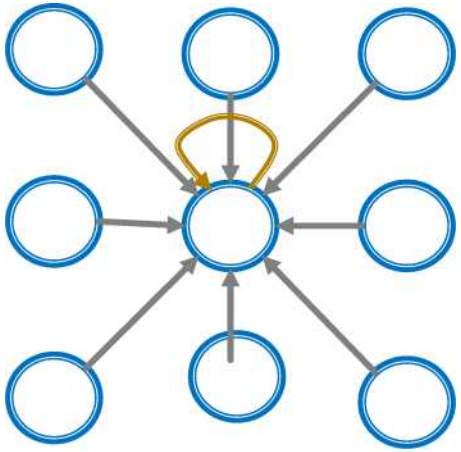
- **Graph $\boldsymbol{G}$:**
- V is the **vertex set**
  - v: a node in V;
  - $N(v)$: the set of neighbors of v
- $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
- $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ is a matrix of **node features**
  - Social networks: user profile
  - Biological networks: gene functional information
  - When there is no node feature in the graph dataset:
    - Indicator vectors (one-hot encoding of a node)
    - Vector of constant 1: [1, 1, …, 1]

Homophily: birds of a feather flock together

- **Idea**: transform information at the neighbors and combine it:
  - Transform "messages" h_i from neighbors: $W_i h_i$
  - Add them up: $\sum_i W_i h_i$

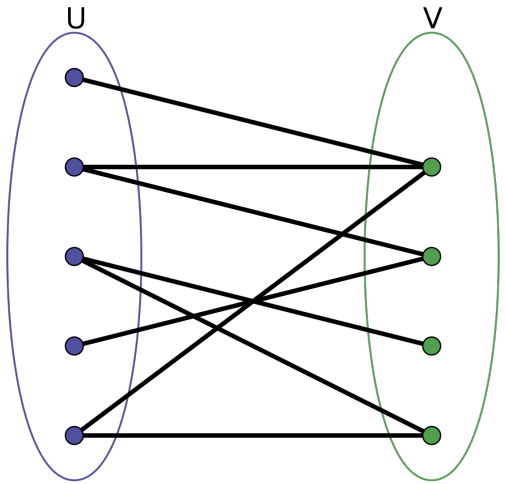


Image



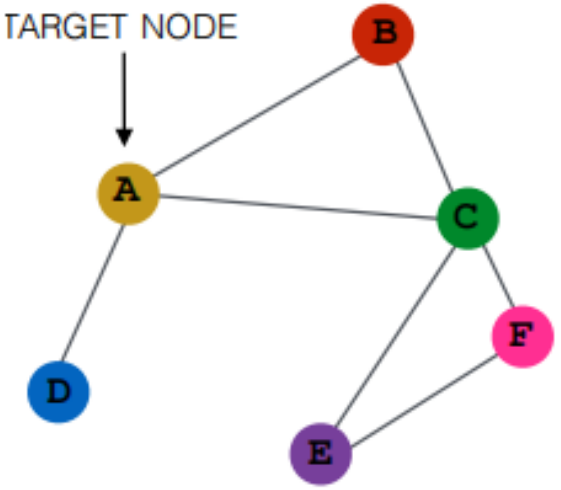
Graph

Example for the least homophily

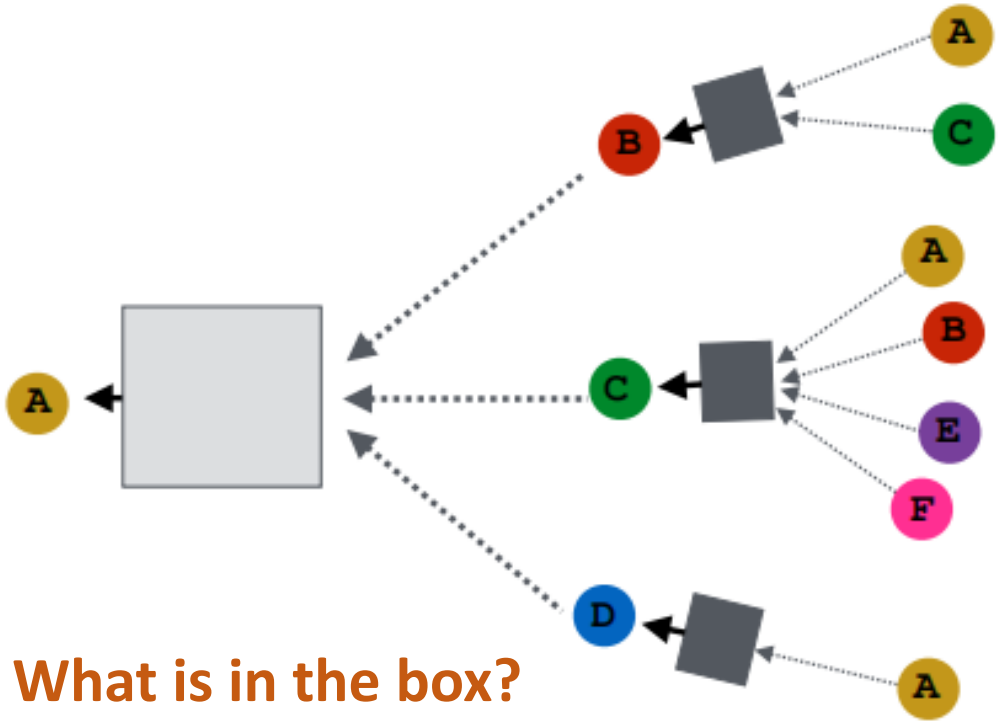


Idea: Aggregate Neighbors

- **Key idea:** Generate node embeddings based on **local network neighborhoods**



INPUT GRAPH

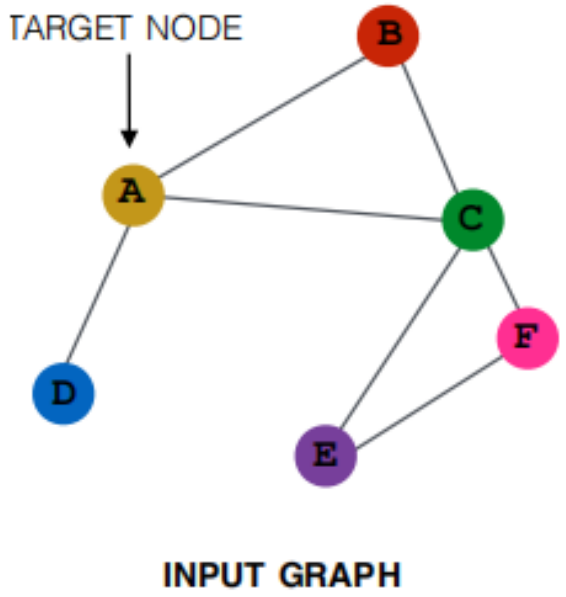


What is in the box?

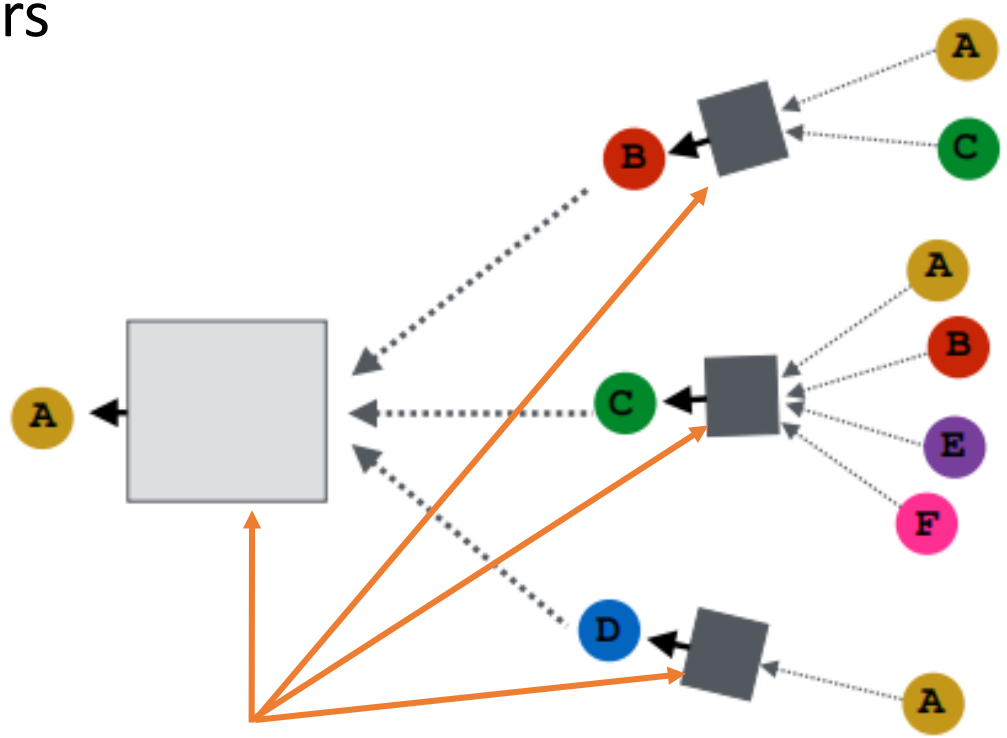
Order insensitive functions: mean, max, some NNs

Graph Neural Networks

- **Intuition:** Nodes aggregate information from their neighbors using neural networks
- Key distinctions of different GNN are in how different approaches aggregate information across the layers



INPUT GRAPH

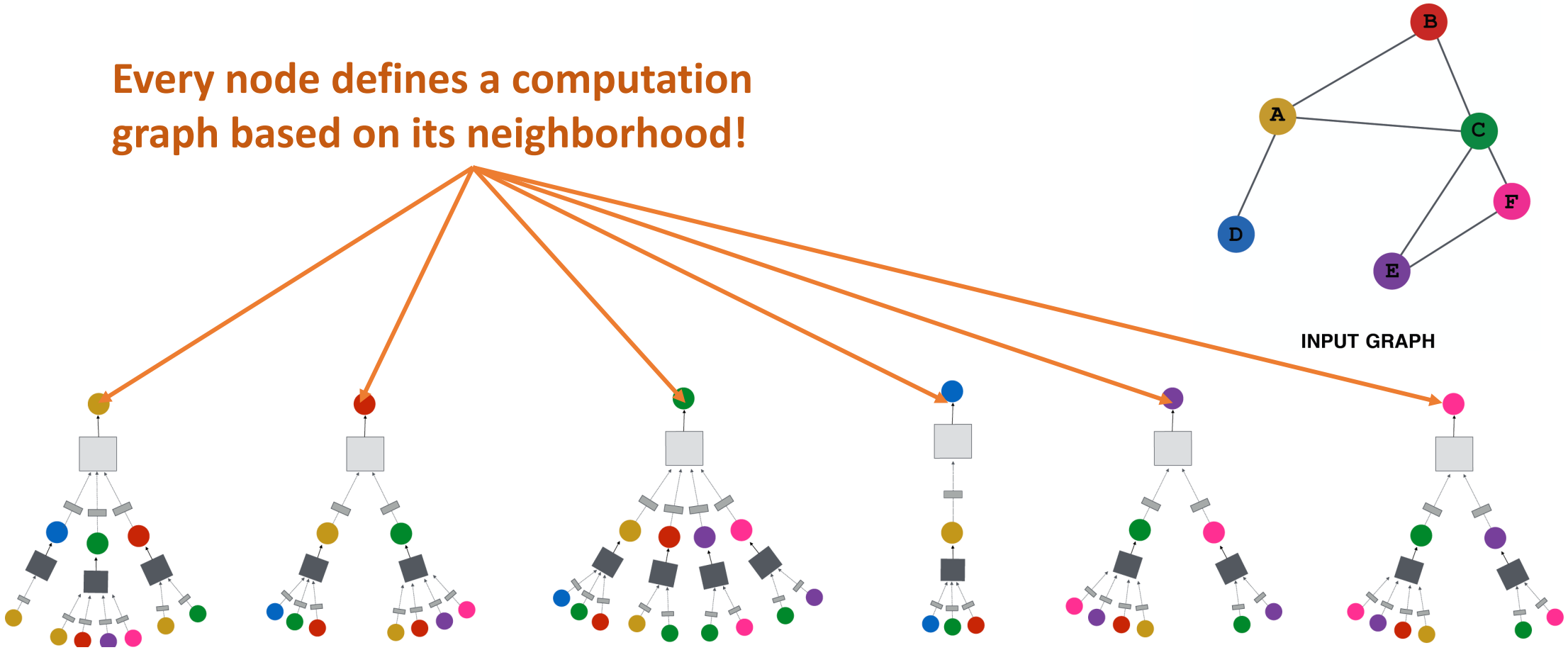


Neural networks

Graph Neural Networks

- **Intuition:** Network neighborhood defines a **computation graph**

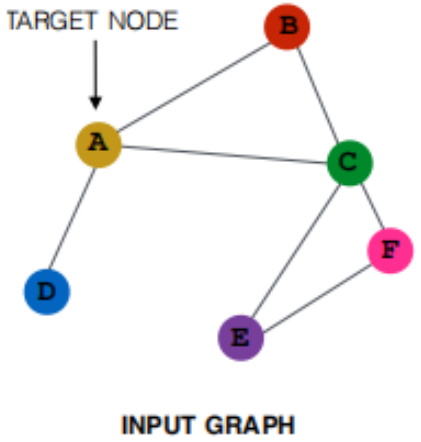
Every node defines a computation graph based on its neighborhood!

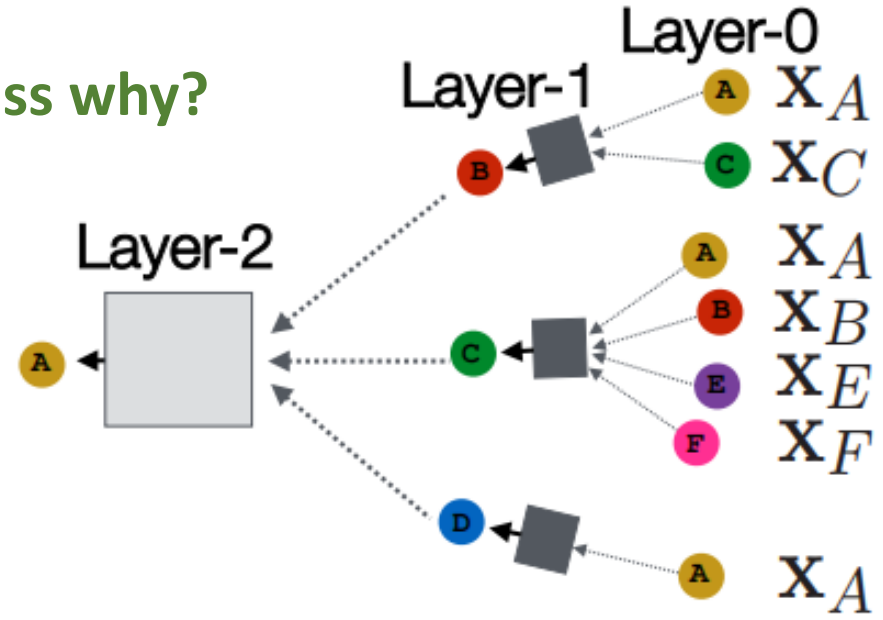


Model Layers

- Model can be **of arbitrary depth**:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node u is its input feature, x_u
  - Layer-k embedding gets information from nodes that are K hops away

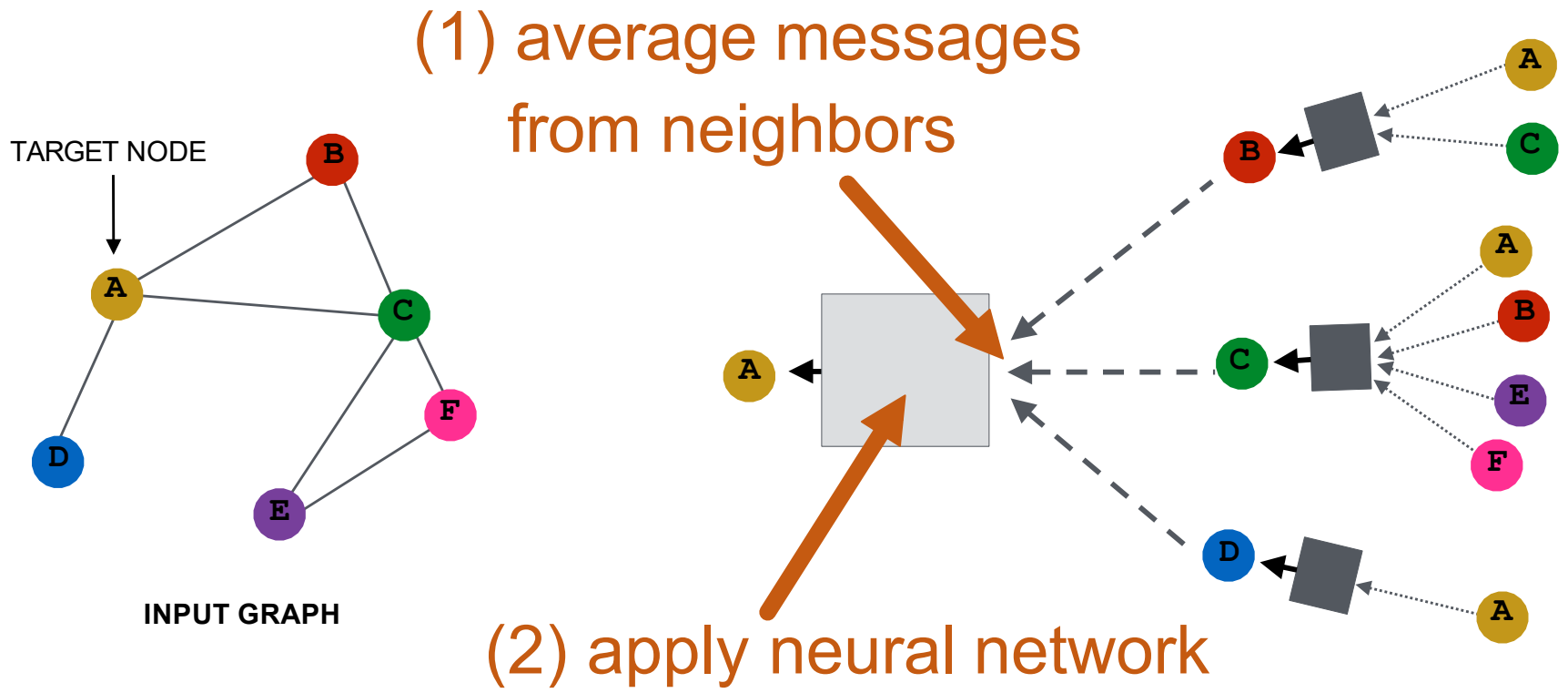
But usually GNN can not be too deep. Guess why?



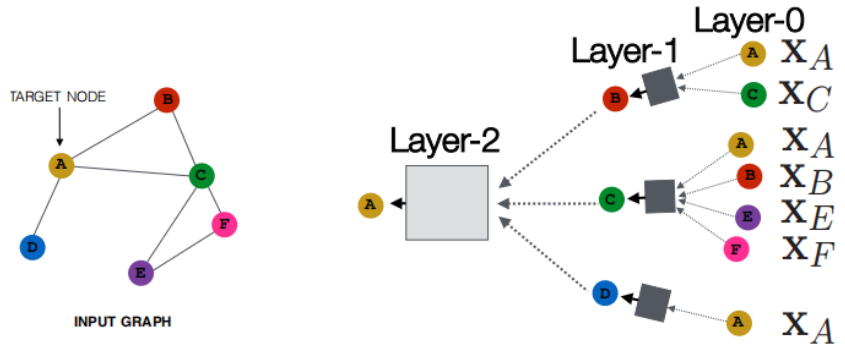


Basic Neighborhood Aggregation

- **Basic approach:** Average information from neighbors and apply a neural network



The Math: Deep Encoder



- **Basic approach:** Average neighbor messages and apply a neural network

Initial 0-th layer embeddings are equal to node features

$$\mathrm{h}_v^0 = \mathrm{x}_v$$

embedding of v at layer l

$$\mathrm{h}_v^{(l+1)} = \sigma\left(\mathrm{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}\right), \forall l \in \{0, \ldots, L-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(L)}$$

Embedding after L-layers of neighborhood aggregation

Non-linearity (e.g., Sigmoid)

Average of neighbor's previous layer embeddings

Total number of layers

Model Parameters

Trainable weight matrices (i.e., what we learn)

$$h_v^{(0)} = x_v$$

$$h_v^{(l+1)} = \sigma\left(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}\right), \forall l \in \{0, \ldots, L-1\}$$

$$z_v = h_v^{(L)}$$

Final node embedding

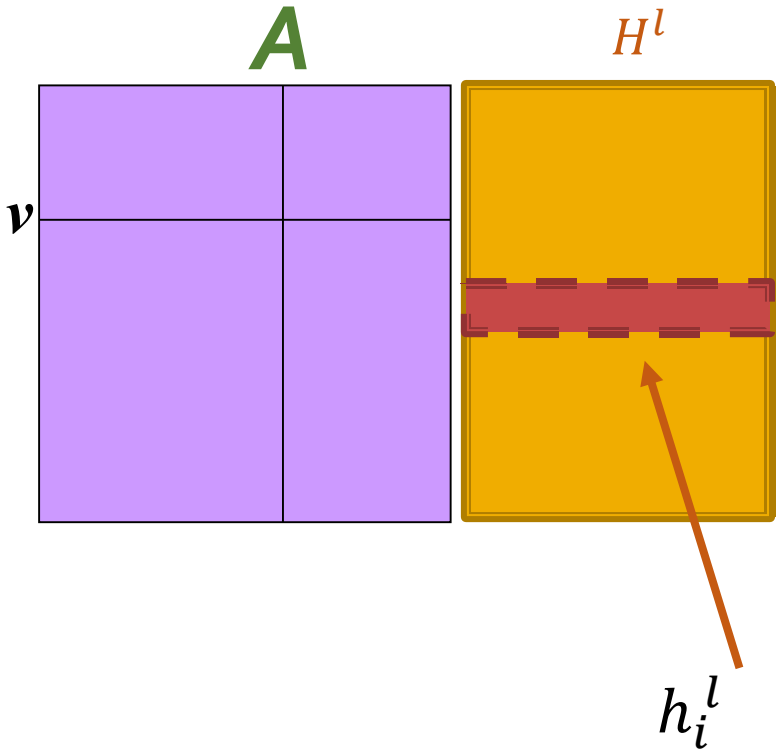
We can feed these **embeddings into any loss function** and run **SGD** to train the **weight** parameters

- h_v^l : the hidden representation of node v at layer l
- W_l: weight matrix for neighborhood aggregation
- B_l: weight matrix for transforming hidden vector of self

Matrix Formulation

- **Many aggregations can be performed efficiently by matrix operations**
- Let $H^{(l)} = \left[h_1^{(l)} \dots h_{|V|}^{(l)}\right]^{\mathrm{T}}$
- Then: $\sum_{u \in N_v} h_u^{(l+1)} = A_{v,:}\mathrm{H}^{(l)}$
- Let D be diagonal matrix where $D_{v,v} = \mathrm{Deg}(v) = |N(v)|$
  - The inverse of D: D^{-1} is also diagonal: $D_{v,v}^{-1} = 1/|N(v)|$
- Therefore,

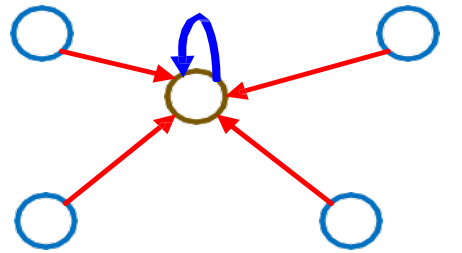
$$\sum_{u \in N(v)} \frac{h_u^l}{|N(v)|} \longrightarrow H^{(l+1)} = D^{-1}AH^{(l)}$$



Matrix Formulation

- Re-writing update function in matrix form:
- $H^{(l+1)} = \sigma\left(\tilde{A}H^{(l)}W_l + H^{(l)}B_l\right)$

  where $\tilde{A} = D^{-1}A$
  - **Red: neighborhood aggregation**
  - **Green: self transformation**



$$H^{(l)} = \left[h_1^{(l)} \dots h_{|V|}^{(l)}\right]^T$$

How to train a GNN

- Node embedding $\boldsymbol{z}_v$ is a function of input graph
- **Supervised setting:** we want to minimize the loss $\mathcal{L}$:

$$\min_{\Theta} \mathcal{L}\big(\boldsymbol{y}, f(\mathbf{z}_v)\big)$$

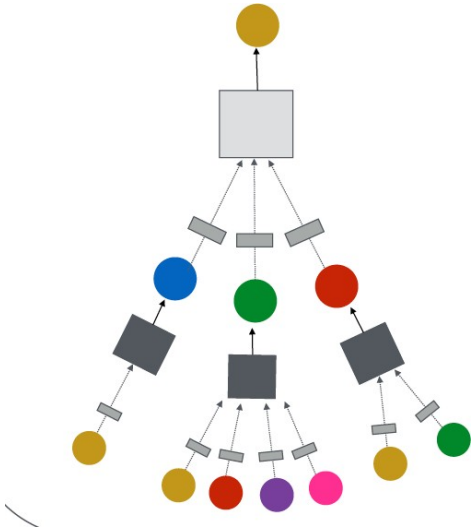
  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical

- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

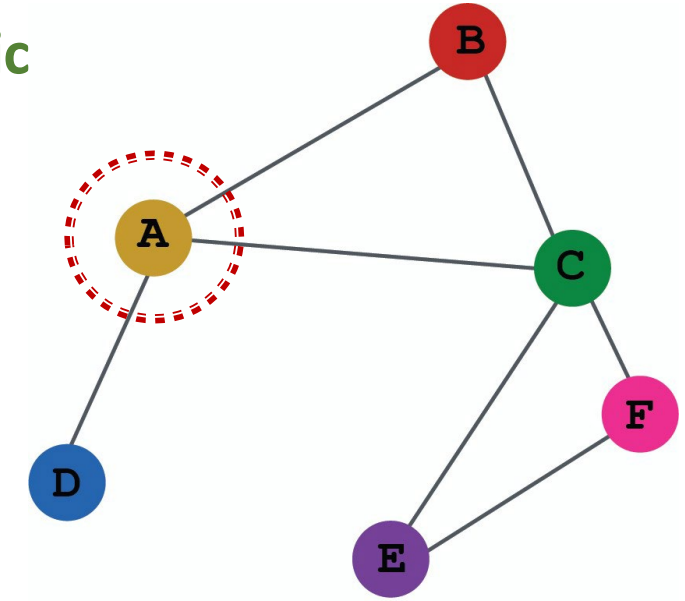
Supervised Training

- **Directly train** the model for a supervised task (e.g., node classification)

Safe or toxic drug?



Safe or toxic drug?



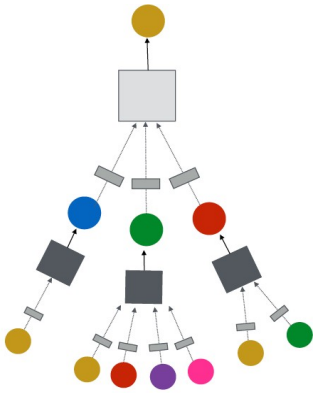
E.g., a drug-drug interaction network

Supervised Training

- **Directly train** the model for a supervised task (e.g., **node classification**)
  - Use cross entropy loss

$$\mathcal{L} = -\left(\sum_{v \in V} y_v \log\left(\sigma\left(\mathbf{z}_v^{\mathrm{T}} \theta\right)\right) + (1 - y_v) \log\left(1 - \sigma\left(\mathbf{z}_v^{\mathrm{T}} \theta\right)\right) \right)$$

Safe or toxic drug?



Encoder output: node embedding

Node class label

Classification weights

Unsupervised Training

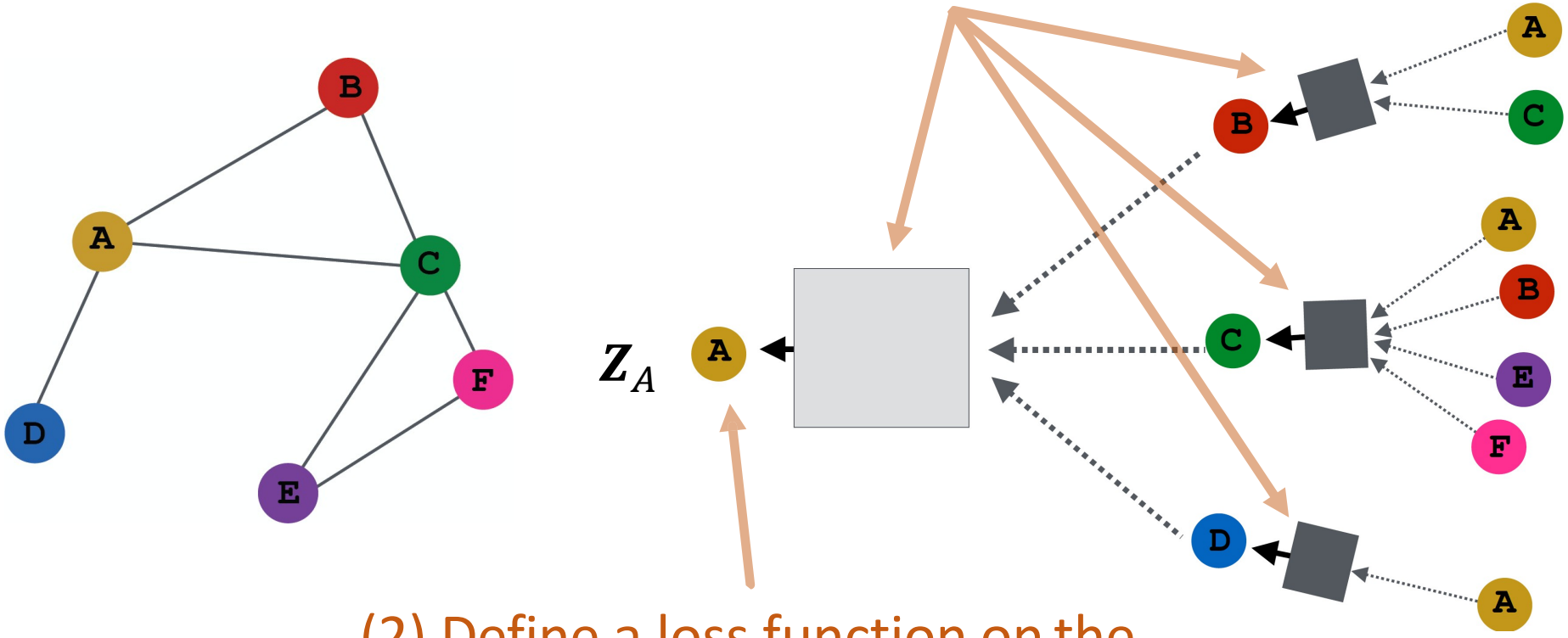
- **"Similar" nodes have similar embeddings**

$$\mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}\left(y_{u,v}, \mathrm{DEC}(z_u, z_v)\right)$$

  - Where $y_{u,v} = 1$ when node u and v are **similar**
  - **CE** is the cross entropy
  - **DEC** is the decoder such as inner product
  - **Node similarity** can be a loss based on:
    - **Random walks** (node2vec, DeepWalk, struc2vec)
    - **Node proximity in the graph**

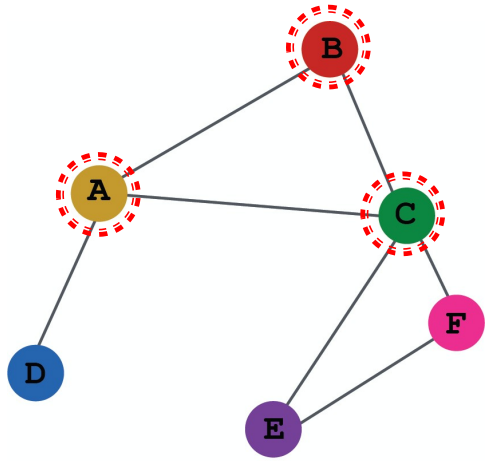
Model Design: Overview

(1) Define a neighborhood aggregation function

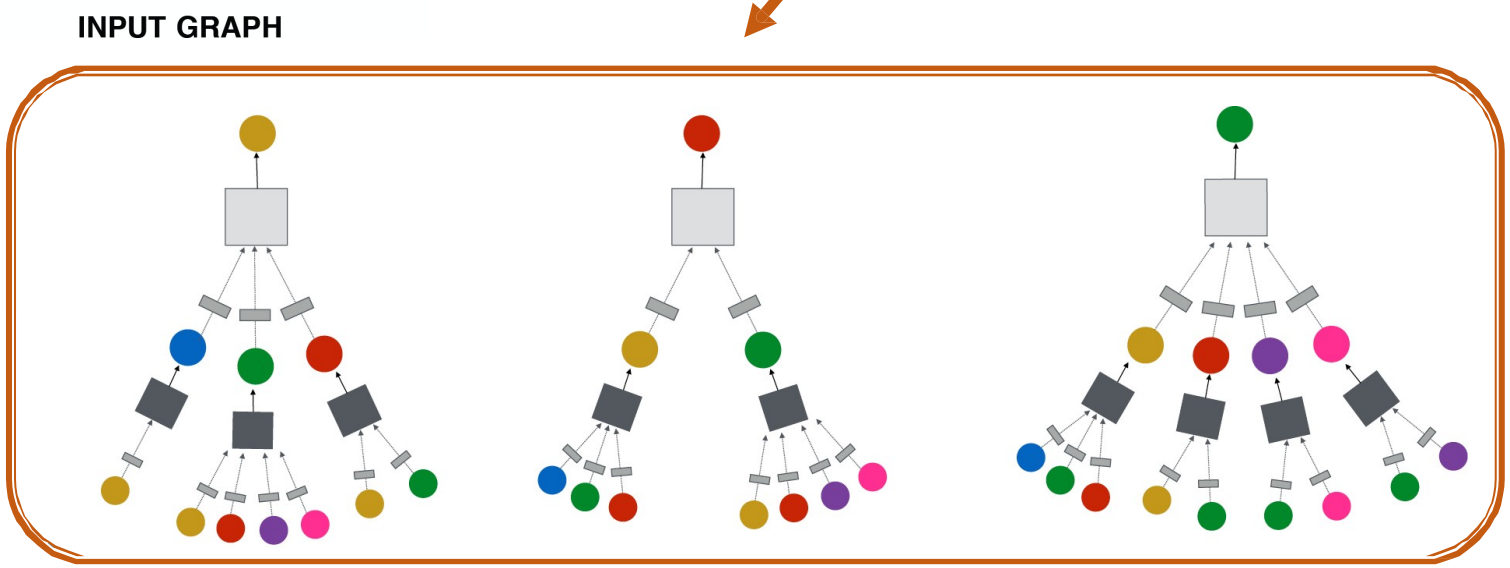


(2) Define a loss function on the embeddings

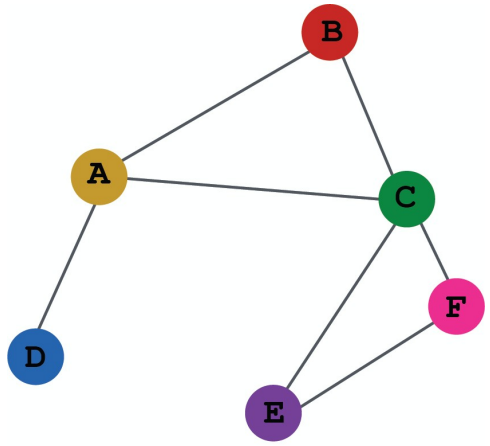
Model Design: Overview



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



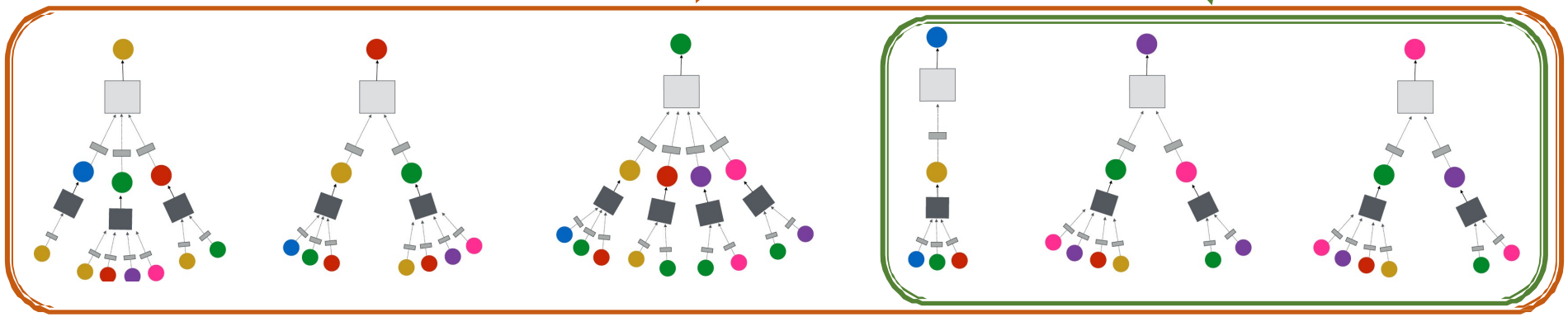
Model Design: Overview



INPUT GRAPH

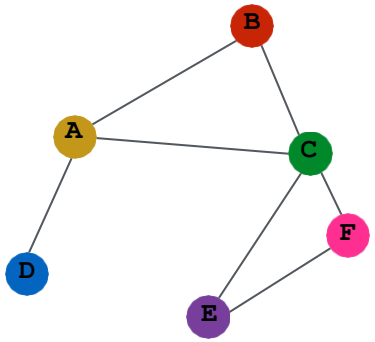
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

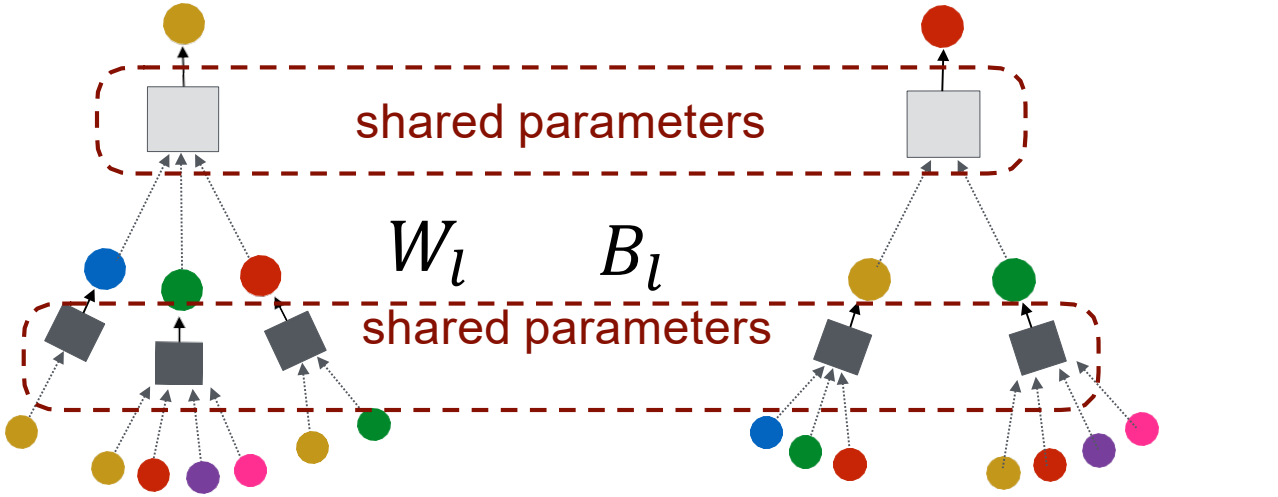


Inductive Capability

- **The same aggregation parameters are shared for all nodes:**
  - The number of model parameters is sublinear in $|V|$ and we can **generalize to unseen nodes**!



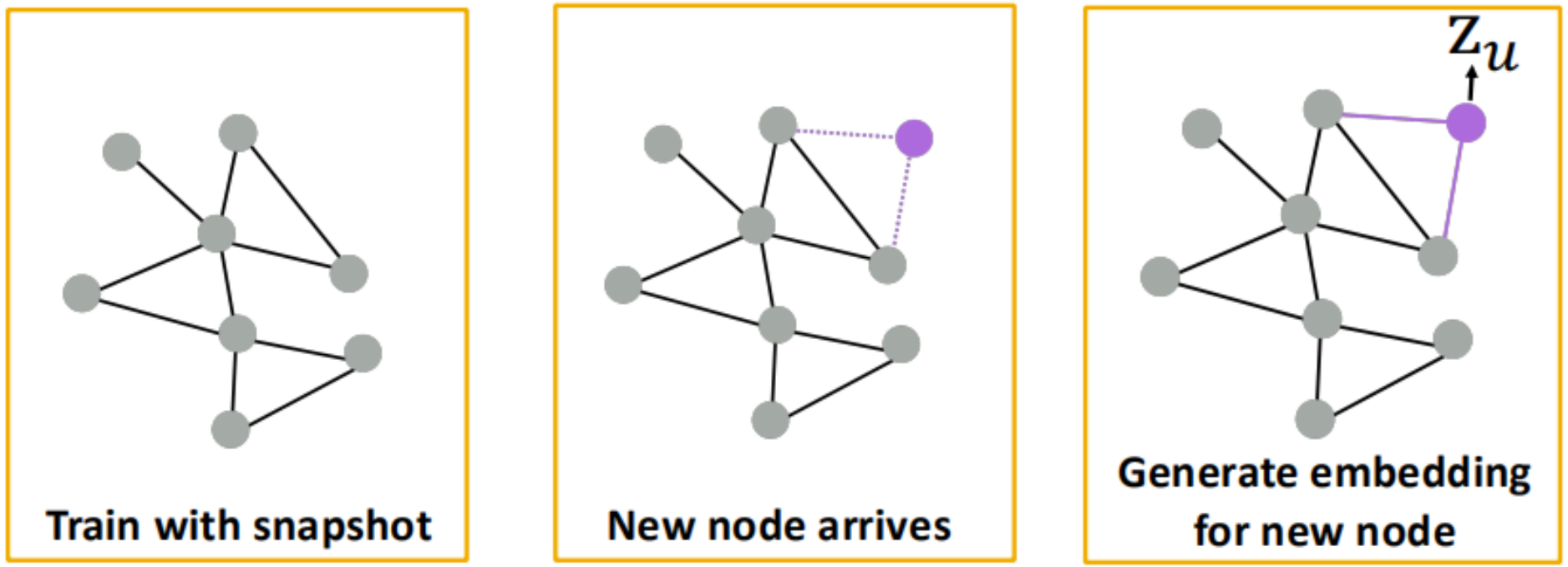
INPUT GRAPH



Compute graph for node A

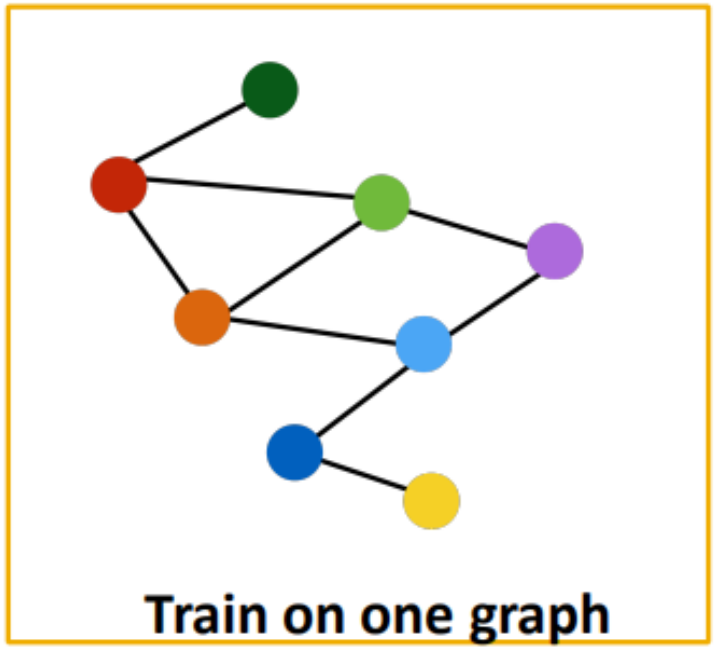
Compute graph for node B

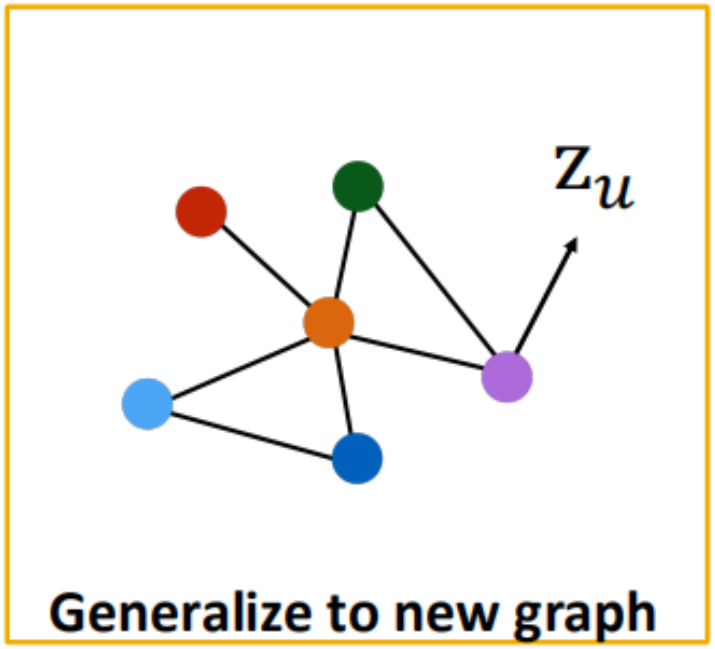
Inductive Capability: New Graphs



- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

Inductive Capability: New Graphs





Inductive node embedding ⟶ Generalize to entirely unseen graphs

E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B