

# Graph Neural Networks - Learning

Acknowledgement: Jure Leskovec, Stanford University

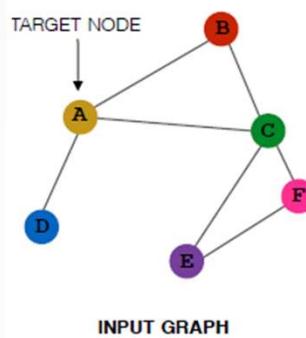
# Overview

- Predicting with GNNs
- Training Graph Neural Networks
- Setting up GNN Prediction Tasks

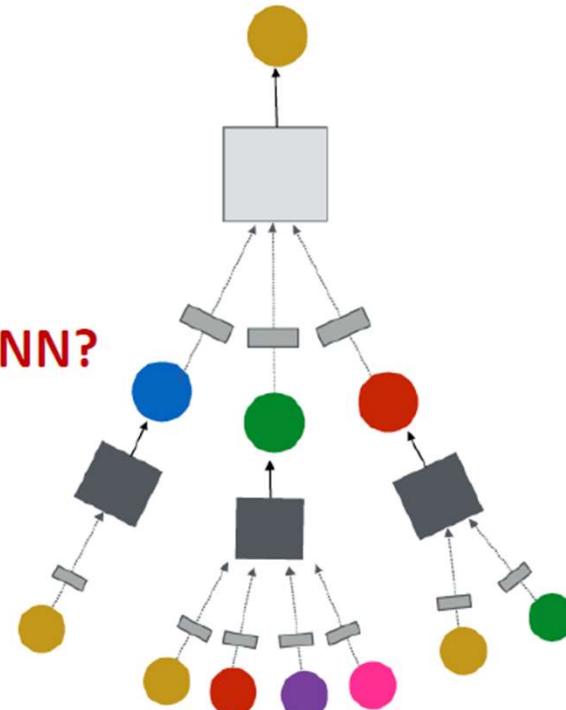
# Overview

- Predicting with GNNs
- Training Graph Neural Networks
- Setting up GNN Prediction Tasks

# A General GNN Framework



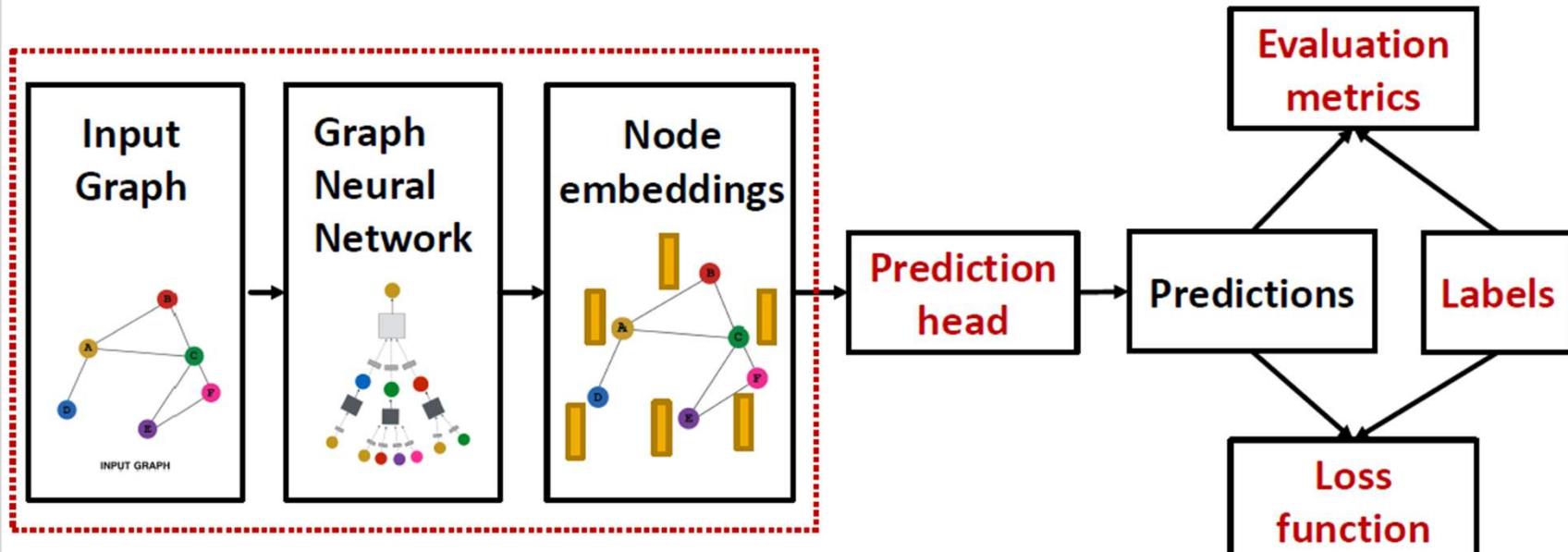
(5) Learning objective



Next: How do we train a GNN?

# GNN Training Pipeline

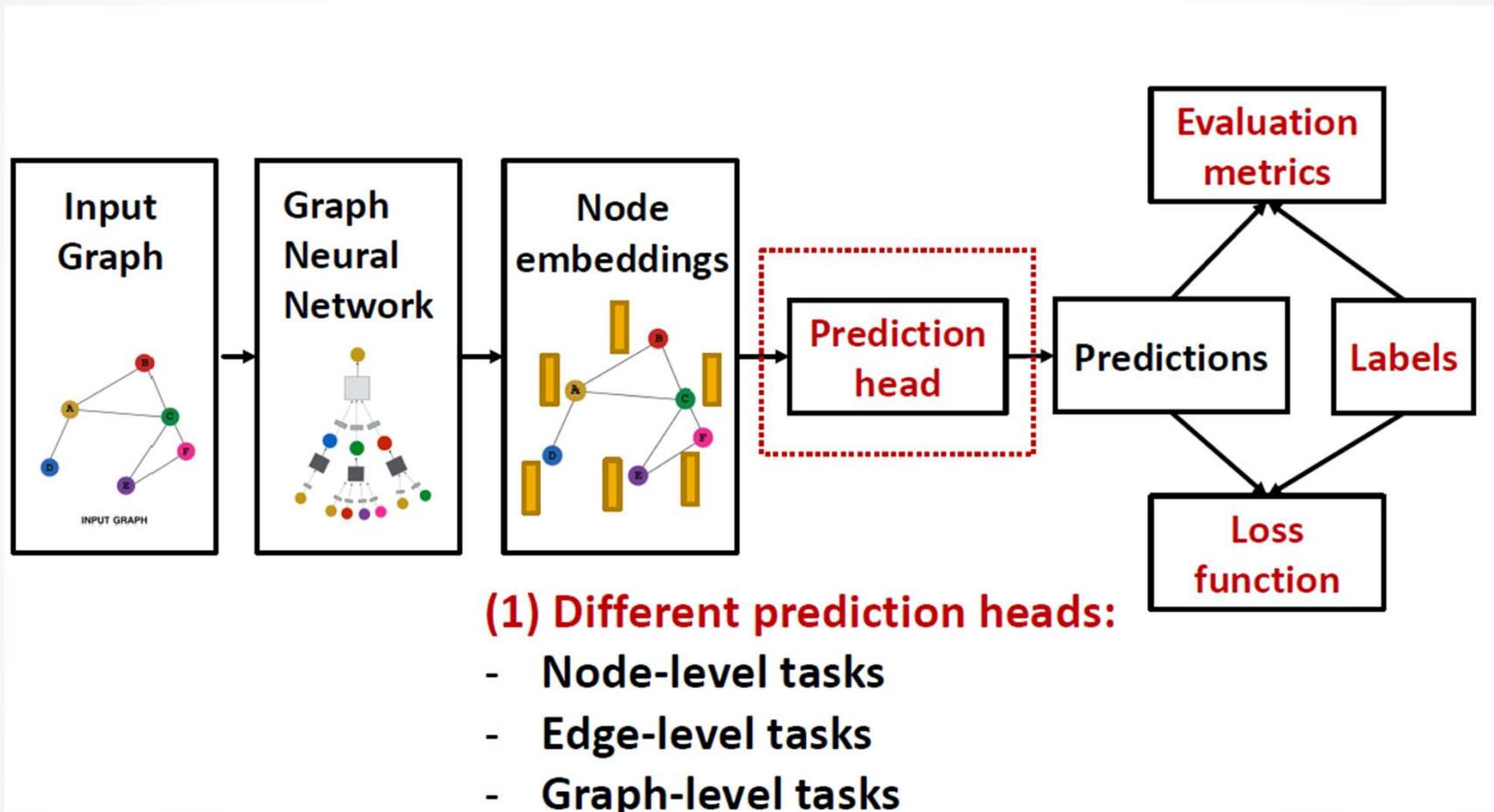
So far what we have covered



Output of a GNN: set of node embeddings

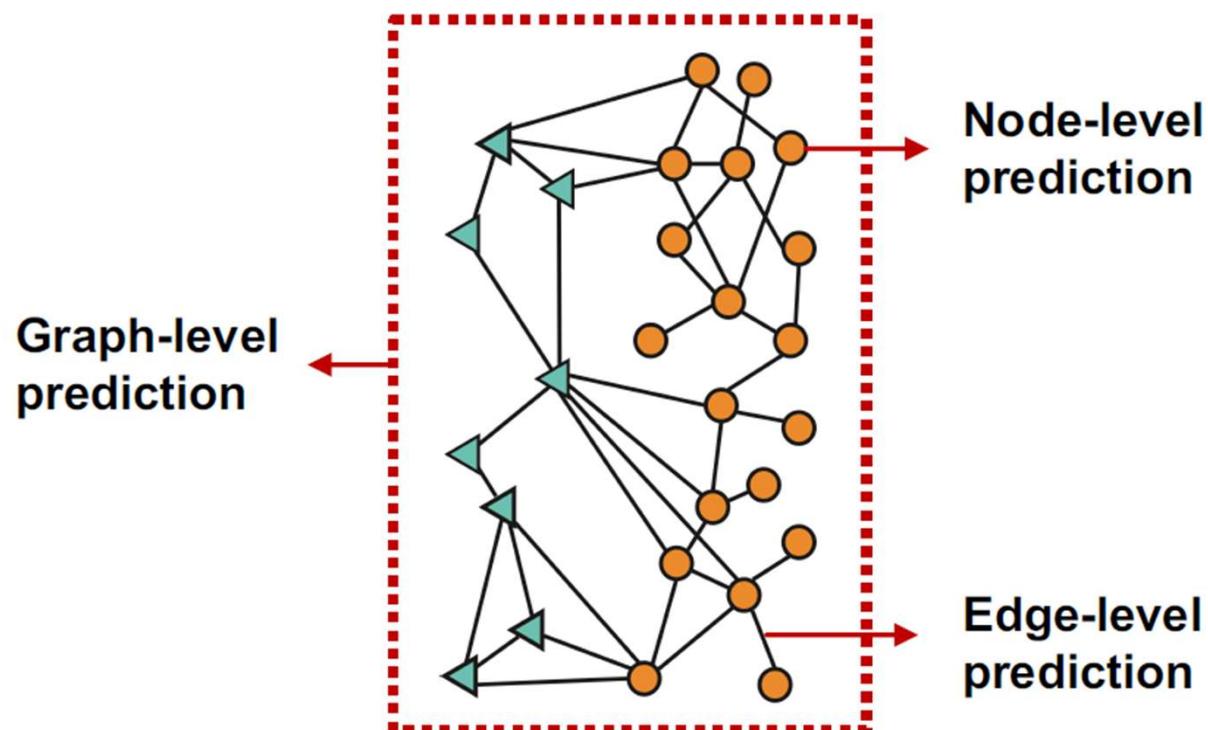
$$\{\mathbf{h}_v^{(L)}, \forall v \in G\}$$

# GNN Training Pipeline



# GNN Prediction Heads

- **Idea:** Different task levels require different prediction heads

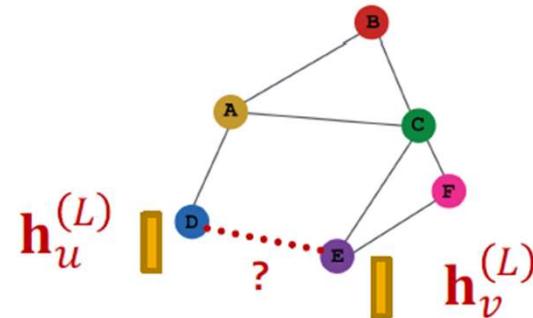


# Prediction Heads: Node level

- **Node-level prediction:** We can directly make prediction using node embeddings!
- After GNN computation, we have  **$d$ -dim node embeddings:**  $\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}$
- Suppose we want to make  **$k$ -way prediction**
  - Classification: classify among  $k$  categories
  - Regression: regress on  $k$  targets
- $\hat{\mathbf{y}}_v = \text{Head}_{\text{node}}(\mathbf{h}_v^{(L)}) = \mathbf{W}^{(H)} \mathbf{h}_v^{(L)}$ 
  - $\mathbf{W}^{(H)} \in \mathbb{R}^{k*d}$ : We map node embeddings from  $\mathbf{h}_v^{(L)} \in \mathbb{R}^d$  to  $\hat{\mathbf{y}}_v \in \mathbb{R}^k$  so that we can compute the loss

# Prediction Heads: Edge level

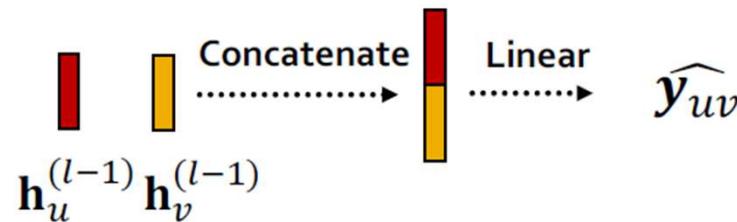
- **Edge-level prediction:** Make prediction using pairs of node embeddings
- Suppose we want to make  $k$ -way prediction
- $\hat{y}_{uv} = \text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$



- What are the options for  $\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$ ?

# Prediction Heads: Edge level

- Options for  $\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$ :
- (1) Concatenation + Linear
  - We have seen this in graph attention



- $\hat{y}_{uv} = \text{Linear}(\text{Concat}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}))$
- Here  $\text{Linear}(\cdot)$  will map  **$2d$ -dimensional** embeddings (since we concatenated embeddings) to  **$k$ -dim** embeddings ( $k$ -way prediction)

# Prediction Heads: Edge level

- Options for  $\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$ :
- (2) Dot product
  - $\hat{y}_{uv} = (\mathbf{h}_u^{(L)})^T \mathbf{h}_v^{(L)}$
  - This approach only applies to 1-way prediction (e.g., link prediction: predict the existence of an edge)
  - Applying to  $k$ -way prediction:
    - Similar to multi-head attention:  $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(k)}$  trainable
$$\hat{y}_{uv}^{(1)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(1)} \mathbf{h}_v^{(L)}$$
$$\hat{y}_{uv}^{(k)} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(k)} \mathbf{h}_v^{(L)}$$
$$\hat{y}_{uv} = \text{Concat}(\hat{y}_{uv}^{(1)}, \dots, \hat{y}_{uv}^{(k)}) \in \mathbb{R}^k$$

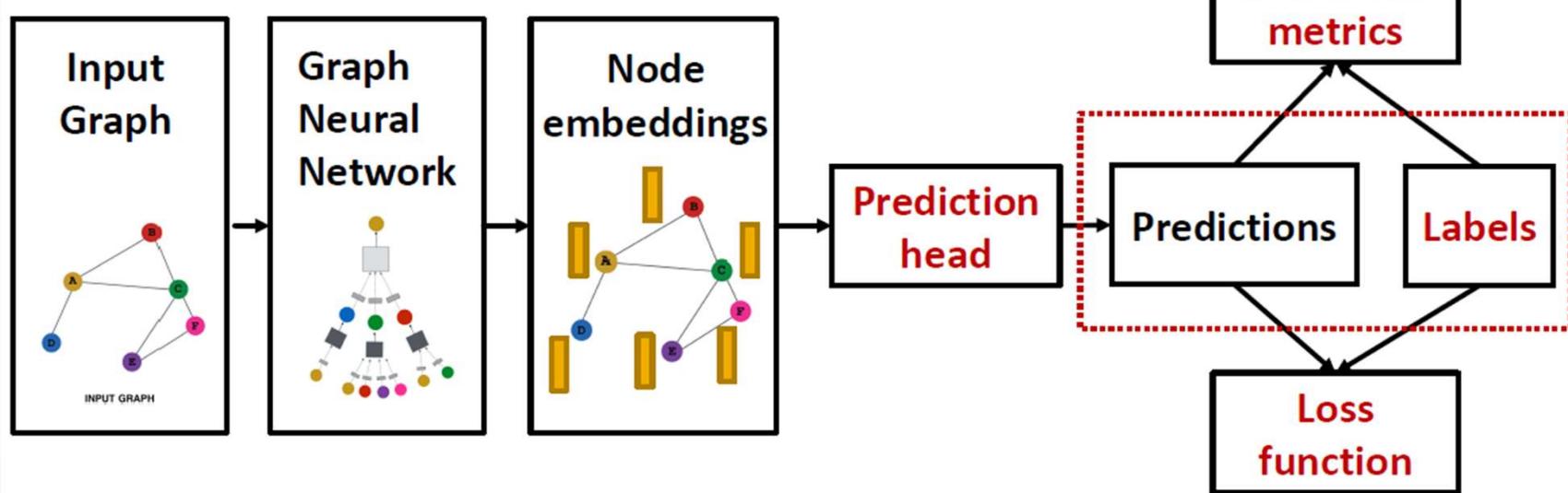
# Overview

- Predicting with GNNs
- Training Graph Neural Networks
- Setting up GNN Prediction Tasks

# GNN Training Pipeline

(2) Where does ground-truth come from?

- Supervised labels
- Unsupervised signals



# Supervised vs Unsupervised

- **Supervised learning on graphs**
  - **Labels come from external sources**
    - E.g., predict drug likeness of a molecular graph
- **Unsupervised learning on graphs**
  - **Signals come from graphs themselves**
    - E.g., link prediction: predict if two nodes are connected
- **Sometimes the differences are blurry**
  - We still have “supervision” in unsupervised learning
    - E.g., train a GNN to predict node clustering coefficient
  - An alternative name for “**unsupervised**” is “**self-supervised**”

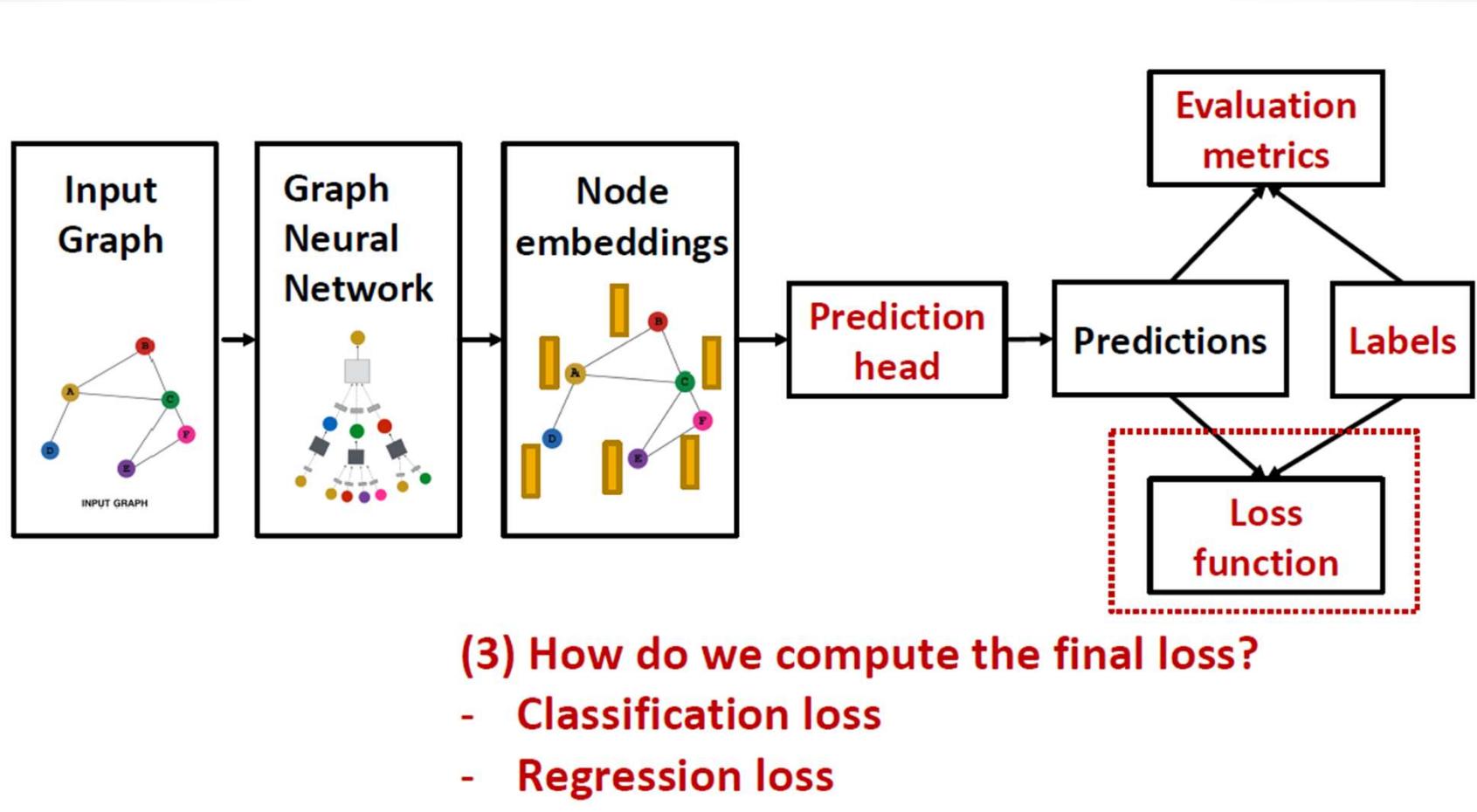
# Supervised Labels on Graphs

- **Supervised labels come from the specific use cases.** For example:
  - **Node labels  $y_v$ :** in a citation network, which subject area does a node belong to
  - **Edge labels  $y_{uv}$ :** in a transaction network, whether an edge is fraudulent
  - **Graph labels  $y_G$ :** among molecular graphs, the drug likeness of graphs
- **Advice:** Reduce your task to node / edge / graph labels, since they are easy to work with
  - E.g., we knew some nodes form a cluster. We can treat the cluster that a node belongs to as a **node label**

# Unsupervised Signals on Graphs

- **The problem:** sometimes we only have a graph, without any external labels
- **The solution:** “self-supervised learning”, we can find supervision signals within the graph.
  - For example, we can let GNN predict the following:
  - **Node-level**  $y_v$ . Node statistics: such as clustering coefficient, PageRank, ...
  - **Edge-level**  $y_{uv}$ . Link prediction: hide the edge between two nodes, predict if there should be a link
  - **Graph-level**  $y_G$ . Graph statistics: for example, predict if two graphs are isomorphic
  - **These tasks do not require any external labels!**

# GNN Training Pipeline



# Settings for GNN Training

- **The setting:** We have  $N$  data points
  - Each data point can be a node/edge/graph
  - **Node-level:** prediction  $\hat{\mathbf{y}}_v^{(i)}$ , label  $\mathbf{y}_v^{(i)}$
  - **Edge-level:** prediction  $\hat{\mathbf{y}}_{uv}^{(i)}$ , label  $\mathbf{y}_{uv}^{(i)}$
  - **Graph-level:** prediction  $\hat{\mathbf{y}}_G^{(i)}$ , label  $\mathbf{y}_G^{(i)}$
  - We will use prediction  $\hat{\mathbf{y}}^{(i)}$ , label  $\mathbf{y}^{(i)}$  to refer **predictions at all levels**

# Classification or Regression

- **Classification:** labels  $y^{(i)}$  with discrete value
  - E.g., Node classification: which category does a node belong to
- **Regression:** labels  $y^{(i)}$  with continuous value
  - E.g., predict the drug likeness of a molecular graph
- GNNs can be applied to both settings
- **Differences: loss function & evaluation metrics**

# Classification Loss

In classification **cross entropy (CE)** is a very common loss function

- *K-way prediction* for  $i$ -th data point:

$$\text{CE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = - \sum_{j=1}^K \mathbf{y}_j^{(i)} \log(\hat{\mathbf{y}}_j^{(i)})$$

Label      Prediction

*i-th data point*  
*j-th class*

where:

E.g. 

0	0	1	0	0
---	---	---	---	---

$\mathbf{y}^{(i)} \in \mathbb{R}^K$  = one-hot label encoding

$\hat{\mathbf{y}}^{(i)} \in \mathbb{R}^K$  = prediction after  $\text{Softmax}(\cdot)$

E.g. 

0.1	0.3	0.4	0.1	0.1
-----	-----	-----	-----	-----

- **Total loss over all  $N$  training examples**

$$\text{Loss} = \sum_{i=1}^N \text{CE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$

# Regression Loss

- For regression tasks we often use **Mean Squared Error (MSE)** a.k.a. **L2 loss**
- K*-way regression for data point (i):

$$\text{MSE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = \sum_{j=1}^K (y_j^{(i)} - \hat{y}_j^{(i)})^2$$

*i*-th data point  
*j*-th target

where:

E.g. 1.4 2.3 1.0 0.5 0.6

$\mathbf{y}^{(i)} \in \mathbb{R}^k$  = Real valued vector of targets

$\hat{\mathbf{y}}^{(i)} \in \mathbb{R}^k$  = Real valued vector of predictions

E.g. 0.9 2.8 2.0 0.3 0.8

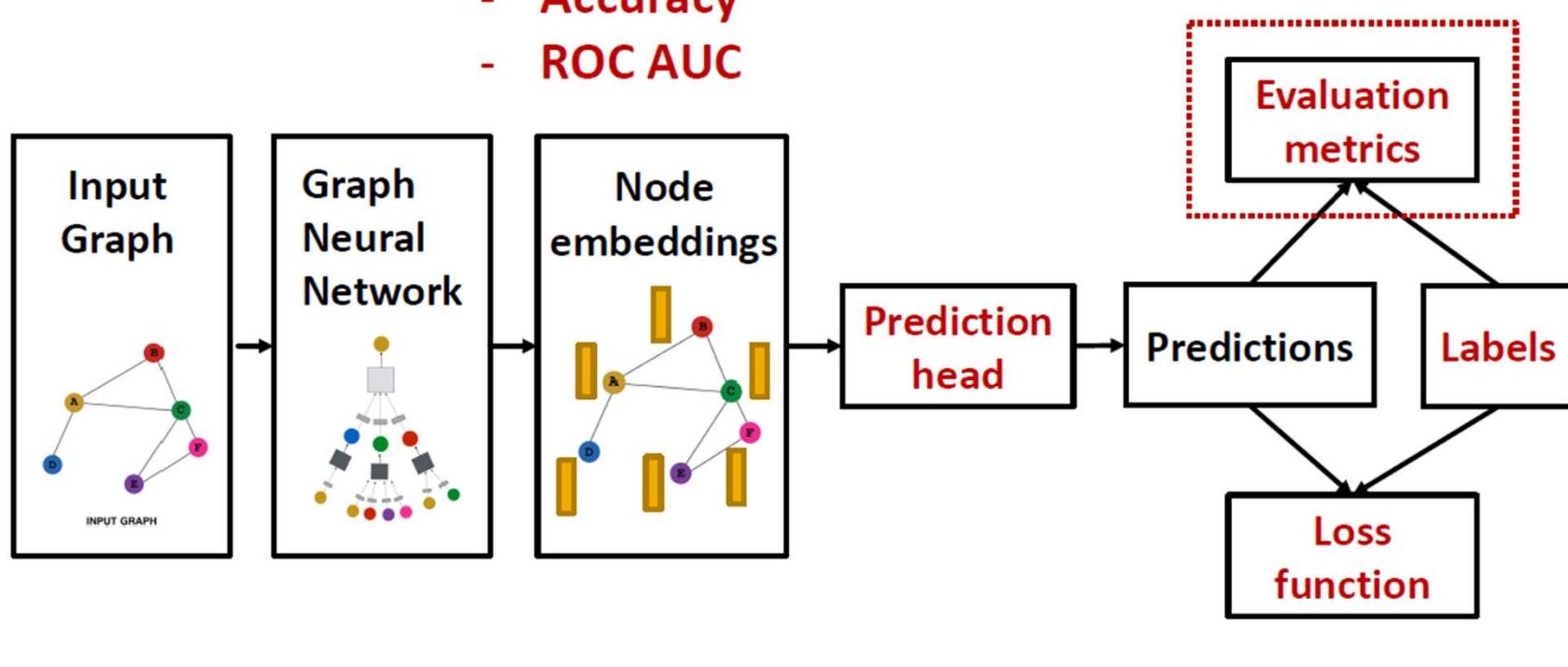
- Total loss over all  $N$  training examples

$$\text{Loss} = \sum_{i=1}^N \text{MSE}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$

# GNN Training Pipeline

(4) How do we measure the success of a GNN?

- Accuracy
- ROC AUC



# Evaluation Metrics: Regression

- We use standard evaluation metrics for GNN
  - (Content below can be found in any ML course)
  - In practice we will use [sklearn](#) for implementation
  - Suppose we make predictions for  $N$  data points
- Evaluate regression tasks on graphs:
  - Root mean square error (RMSE)

$$\sqrt{\sum_{i=1}^N \frac{(y^{(i)} - \hat{y}^{(i)})^2}{N}}$$

- Mean absolute error (MAE)

$$\frac{\sum_{i=1}^N |y^{(i)} - \hat{y}^{(i)}|}{N}$$

# Evaluation Metrics: Classification

- Evaluate classification tasks on graphs:
- (1) Multi-class classification
  - We simply report the accuracy

$$\frac{1[\operatorname{argmax}(\hat{\mathbf{y}}^{(i)}) = \mathbf{y}^{(i)}]}{N}$$

- (2) Binary classification
  - Metrics sensitive to classification threshold
    - Accuracy
    - Precision / Recall
    - If the range of prediction is [0,1], we will use 0.5 as threshold
  - Metric Agnostic to classification threshold
    - ROC AUC

# Metrics for Binary Classification

- **Accuracy:**

$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{|\text{Dataset}|}$$

- **Precision (P):**

$$\frac{TP}{TP + FP}$$

- **Recall (R):**

$$\frac{TP}{TP + FN}$$

- **F1-Score:**

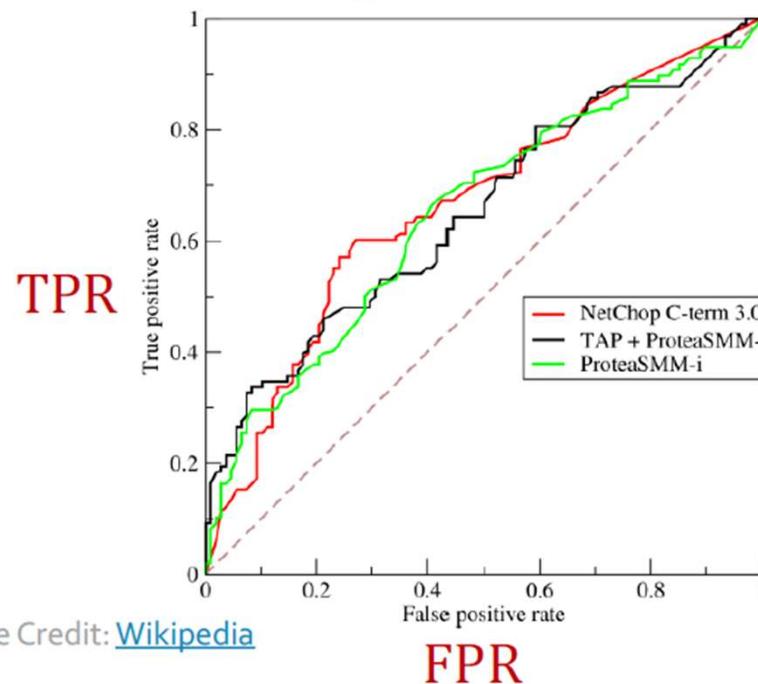
$$\frac{2P * R}{P + R}$$

**Confusion matrix**

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

# Evaluation Metrics

- **ROC Curve:** Captures the tradeoff in TPR and FPR **as the classification threshold is varied for a binary classifier.**

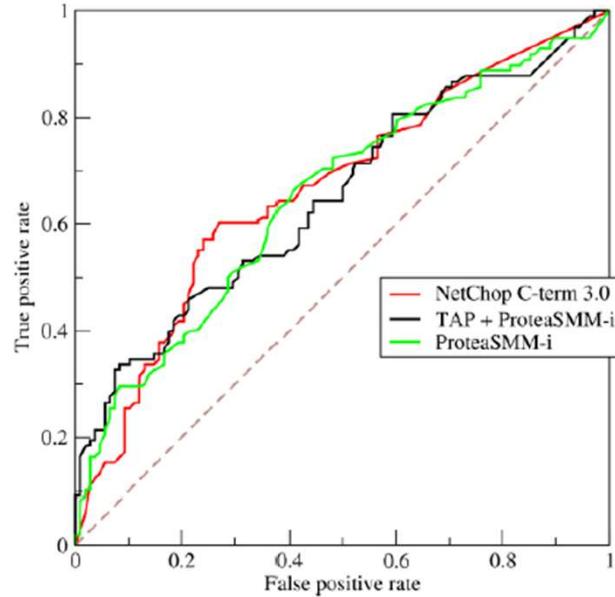


$$\text{TPR} = \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

**Note:** the dashed line represents performance of a random classifier

# Evaluation Metrics



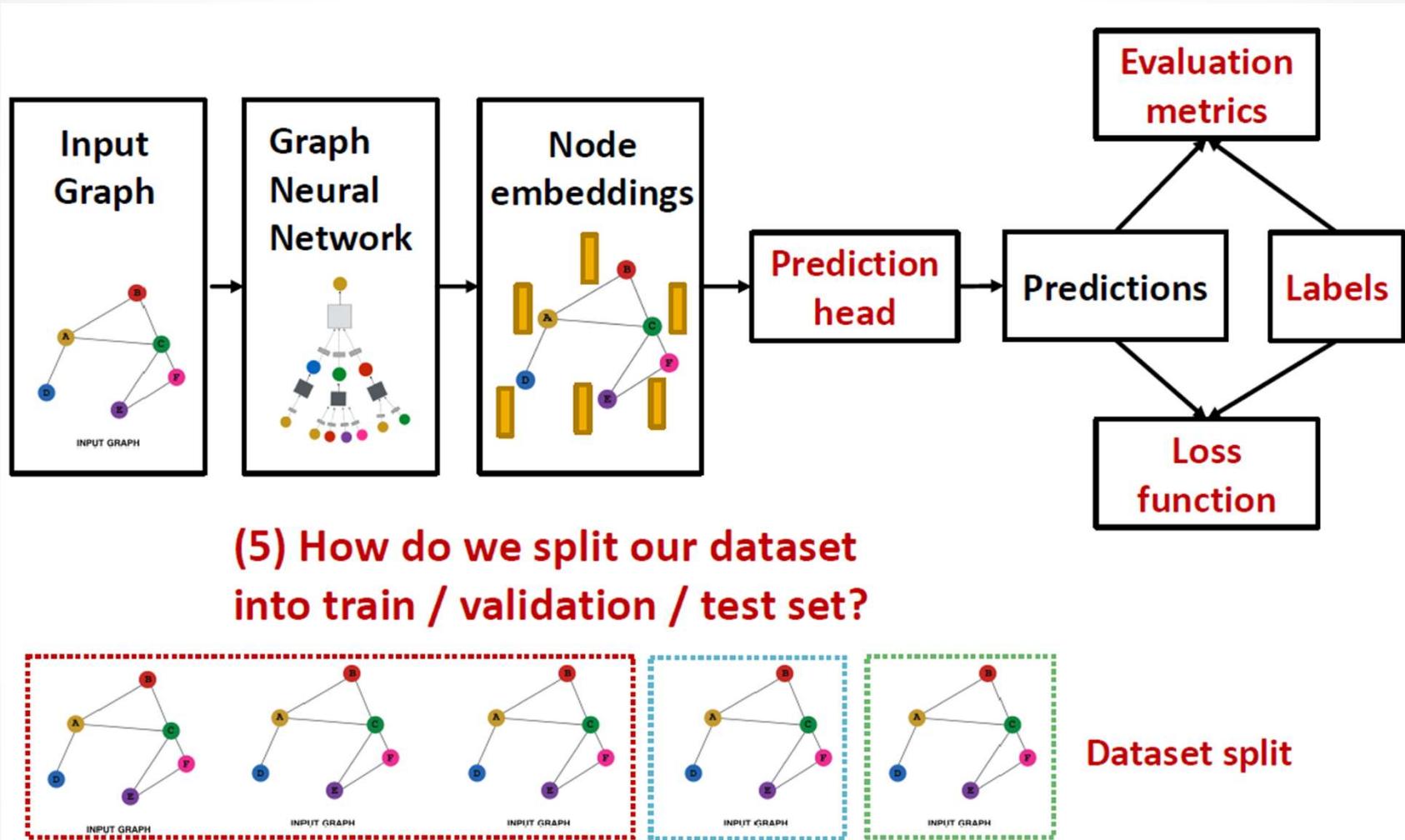
Content Credit: [Wikipedia](#)

- **ROC AUC: Area under the ROC Curve.**
- **Intuition:** The probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one

# Overview

- Predicting with GNNs
- Training Graph Neural Networks
- Setting up GNN Prediction Tasks

# GNN Training Pipeline

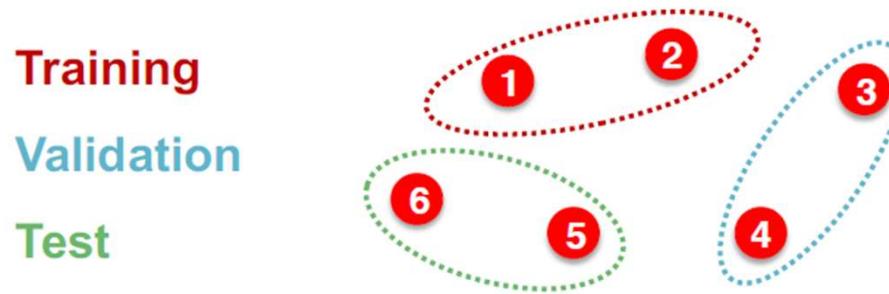


# Dataset Split: Fixed/Random

- **Fixed split:** We will split our dataset **once**
  - **Training set:** used for optimizing GNN parameters
  - **Validation set:** develop model/hyperparameters
  - **Test set:** held out until we report final performance
- **A concern:** sometimes we cannot guarantee that the test set will really be held out
- **Random split:** we will **randomly split** our dataset into training / validation / test
  - We report **average performance over different random seeds**

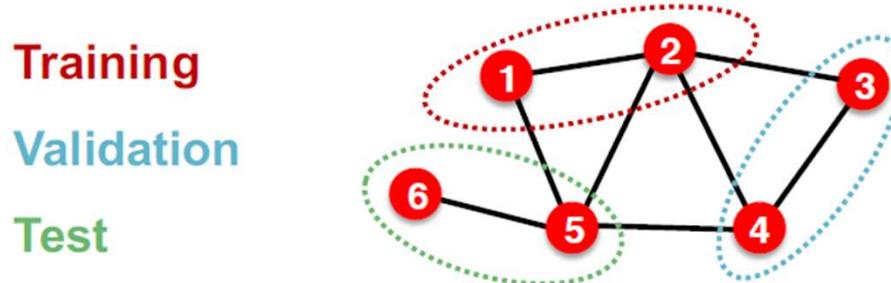
# Why Splitting Graphs is Special

- Suppose we want to split an image dataset
  - **Image classification:** Each data point is an image
  - Here **data points are independent**
    - Image 5 will not affect our prediction on image 1



# Why Splitting Graphs is Special

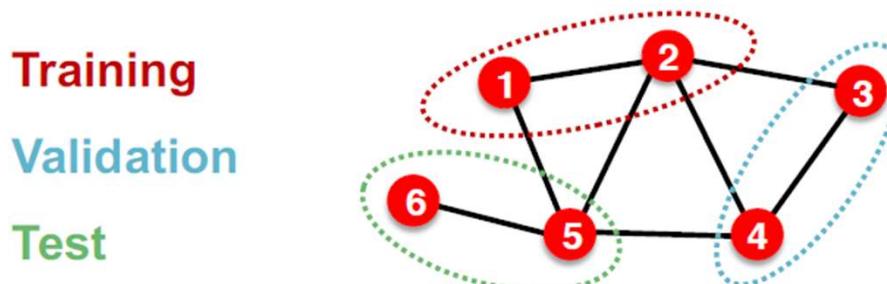
- **Splitting a graph dataset is different!**
  - **Node classification:** Each data point is a node
  - Here **data points are NOT independent**
    - **Node 5 will affect our prediction on node 1**, because it will participate in message passing → affect node 1's embedding



- **What are our options?**

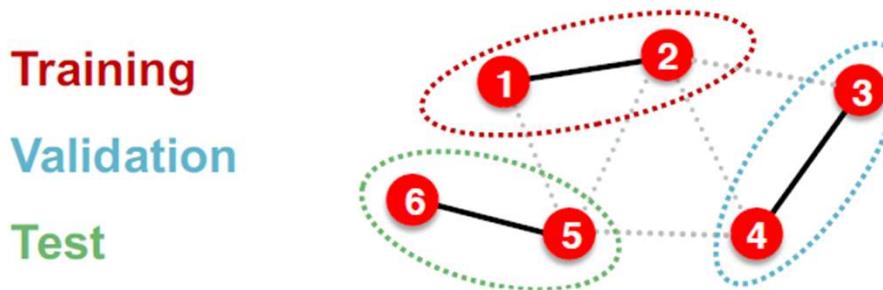
# Why Splitting Graphs is Special

- **Solution 1 (Transductive setting): The input graph can be observed in all the dataset splits (training, validation and test set).**
- **We will only split the (node) labels**
  - At training time, we compute embeddings using the entire graph, and train using node 1&2's labels
  - At validation time, we compute embeddings using the entire graph, and evaluate on node 3&4's labels



# Why Splitting Graphs is Special

- **Solution 2 (Inductive setting): We break the edges between splits to get multiple graphs**
  - Now we have 3 graphs that are independent. Node 5 will not affect our prediction on node 1 any more
  - At training time, we compute embeddings using the graph over node 1&2, and train using node 1&2's labels
  - At validation time, we compute embeddings using the graph over node 3&4, and evaluate on node 3&4's labels



# Transductive/Inductive Settings

- **Transductive setting:** training / validation / test sets are **on the same graph**
  - The **dataset consists of one graph**
  - **The entire graph can be observed in all dataset splits, we only split the labels**
  - Only applicable to **node / edge** prediction tasks
- **Inductive setting:** training / validation / test sets are **on different graphs**
  - The **dataset consists of multiple graphs**
  - Each split can **only observe the graph(s) within the split.** A successful model should **generalize to unseen graphs**
  - Applicable to **node / edge / graph** tasks

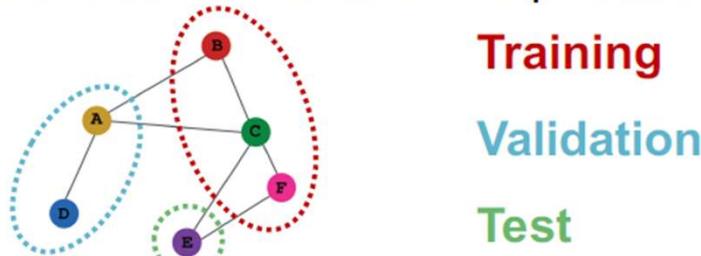
## Disadvantage:

Transductive: There will be information leakage between training, validation and test set

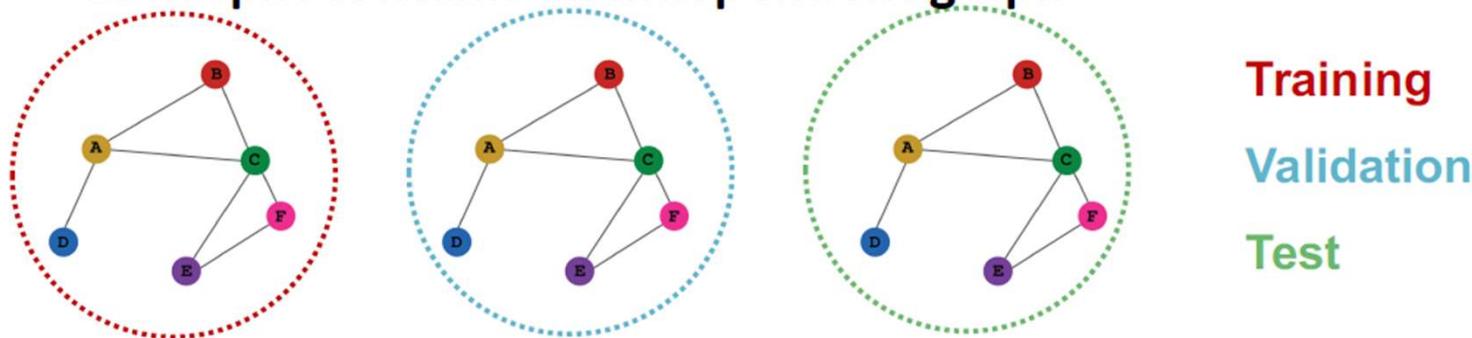
Inductive: Information loss since some edges are removed

# Example: Node Classification

- **Transductive** node classification
  - All the splits can observe the entire graph structure, but can only observe the labels of their respective nodes

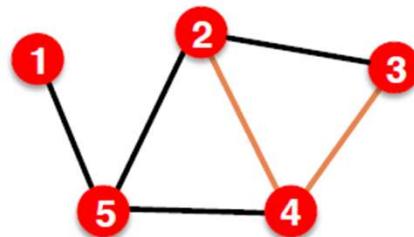


- **Inductive** node classification
  - Suppose we have a dataset of 3 graphs
  - **Each split contains an independent graph**

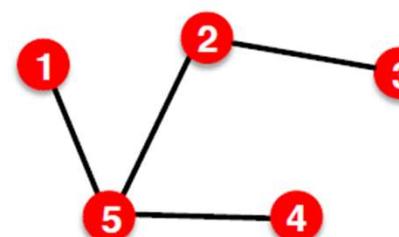


# Example: Link Prediction

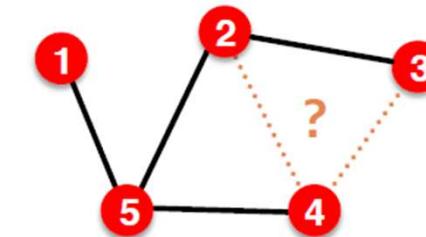
- **Goal of link prediction:** predict missing edges
- **Setting up link prediction is tricky:**
  - Link prediction is an unsupervised / self-supervised task. We need to **create the labels** and **dataset splits** on our own
  - Concretely, we need to **hide some edges** from the **GNN** and let the **GNN** predict if the edges exist



Original graph

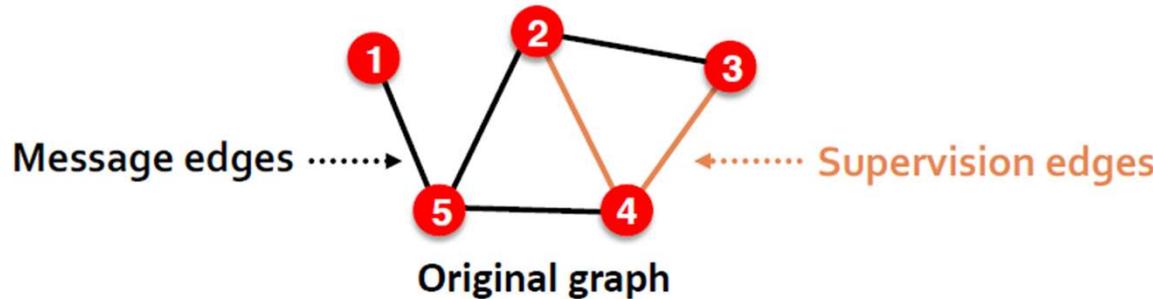


Input graph to GNN



Predictions made by GNN

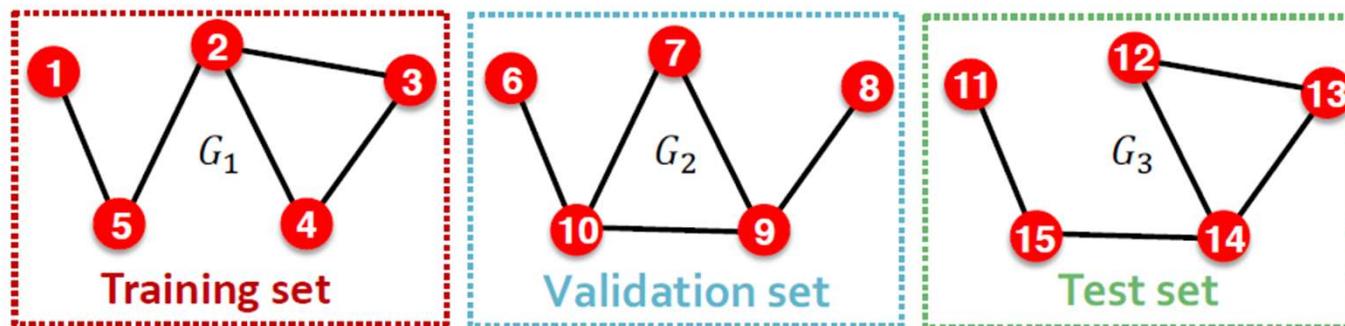
# Setting up Link Prediction



- For link prediction, we will split edges twice
- Step 1: Assign 2 types of edges in the original graph
  - Message edges: Used for GNN message passing
  - Supervision edges: Use for computing objectives
  - After step 1:
    - Only message edges will remain in the graph
    - Supervision edges are used as supervision for edge predictions made by the model, will not be fed into GNN!

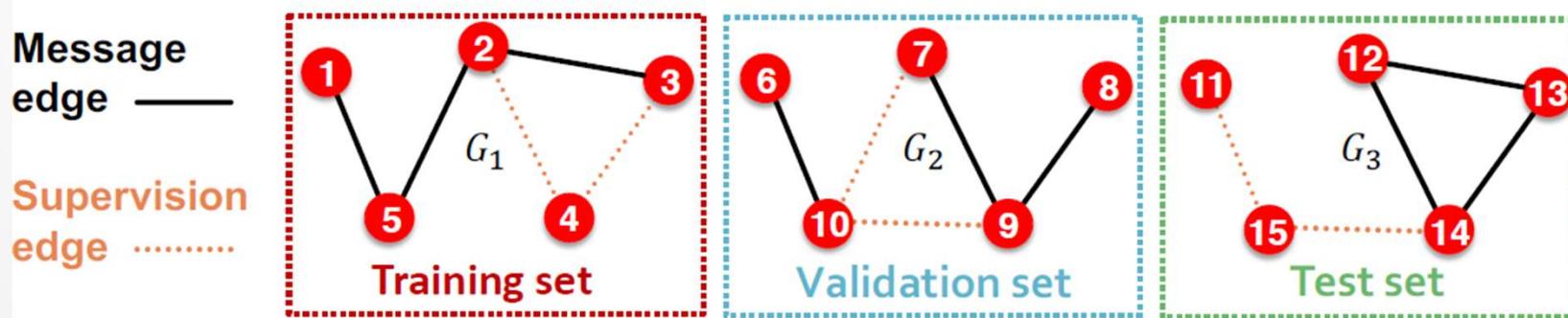
# Setting up Link Prediction

- Step 2: Split edges into train / validation / test
- Option 1: Inductive link prediction split
  - Suppose we have a dataset of 3 graphs. Each inductive split will contain an independent graph



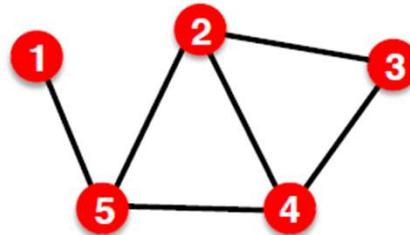
# Setting up Link Prediction

- Step 2: Split edges into train / validation / test
- Option 1: Inductive link prediction split
  - Suppose we have a dataset of 3 graphs. Each inductive split will contain an independent graph
  - In **train** or **val** or **test** set, each graph will have **2 types of edges: message edges + supervision edges**
    - **Supervision edges** are not the input to GNN



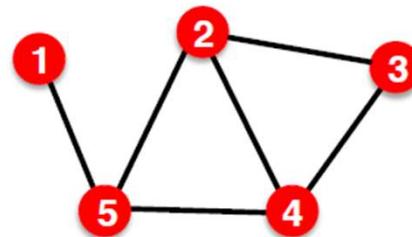
# Setting up Link Prediction

- **Option 2: Transductive link prediction split:**
  - This is the default setting when people talk about link prediction
  - Suppose we have a dataset of 1 graph



# Setting up Link Prediction

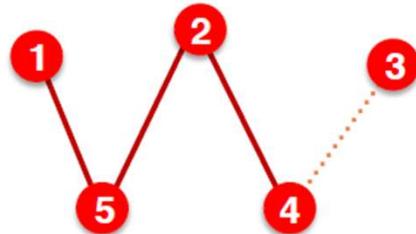
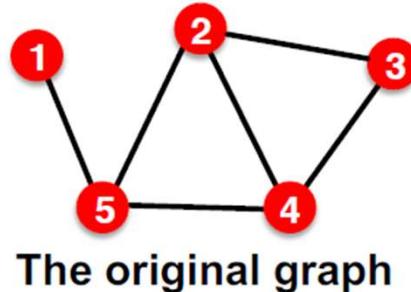
- **Option 2: Transductive link prediction split:**
  - By definition of “transductive”, the entire graph can be observed in all dataset splits
    - But since edges are both part of graph structure and the supervision, we need to hold out validation / test edges
    - To train the **training** set, we further need to hold out supervision edges for the **training** set



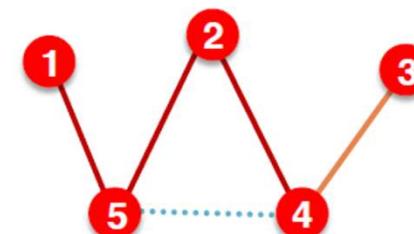
- **Next:** we will show the exact settings

# Setting up Link Prediction

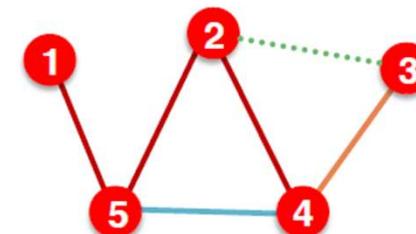
## ■ Option 2: Transductive link prediction split:



**(1) At training time:**  
Use **training message edges** to predict **training supervision edges**



**(2) At validation time:**  
Use **training message edges & training supervision edges** to predict **validation edges**



**(3) At test time:**  
Use **training message edges & training supervision edges & validation edges** to predict **test edges**

# Setting up Link Prediction

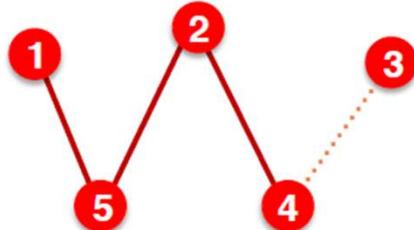
## ■ Option 2: Transductive link prediction split:

Why do we use growing number of edges?

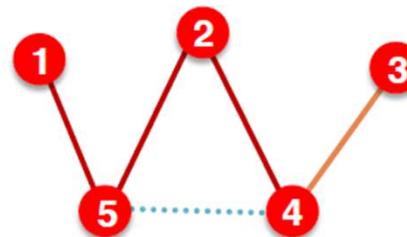
After training, supervision edges are known to GNN.

Therefore, an ideal model should use supervision edges in message passing at validation time.

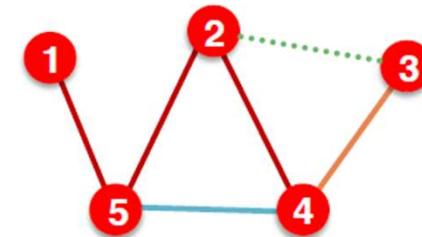
The same applies to the test time.



(1) At training time:  
Use **training message edges** to predict **training supervision edges**

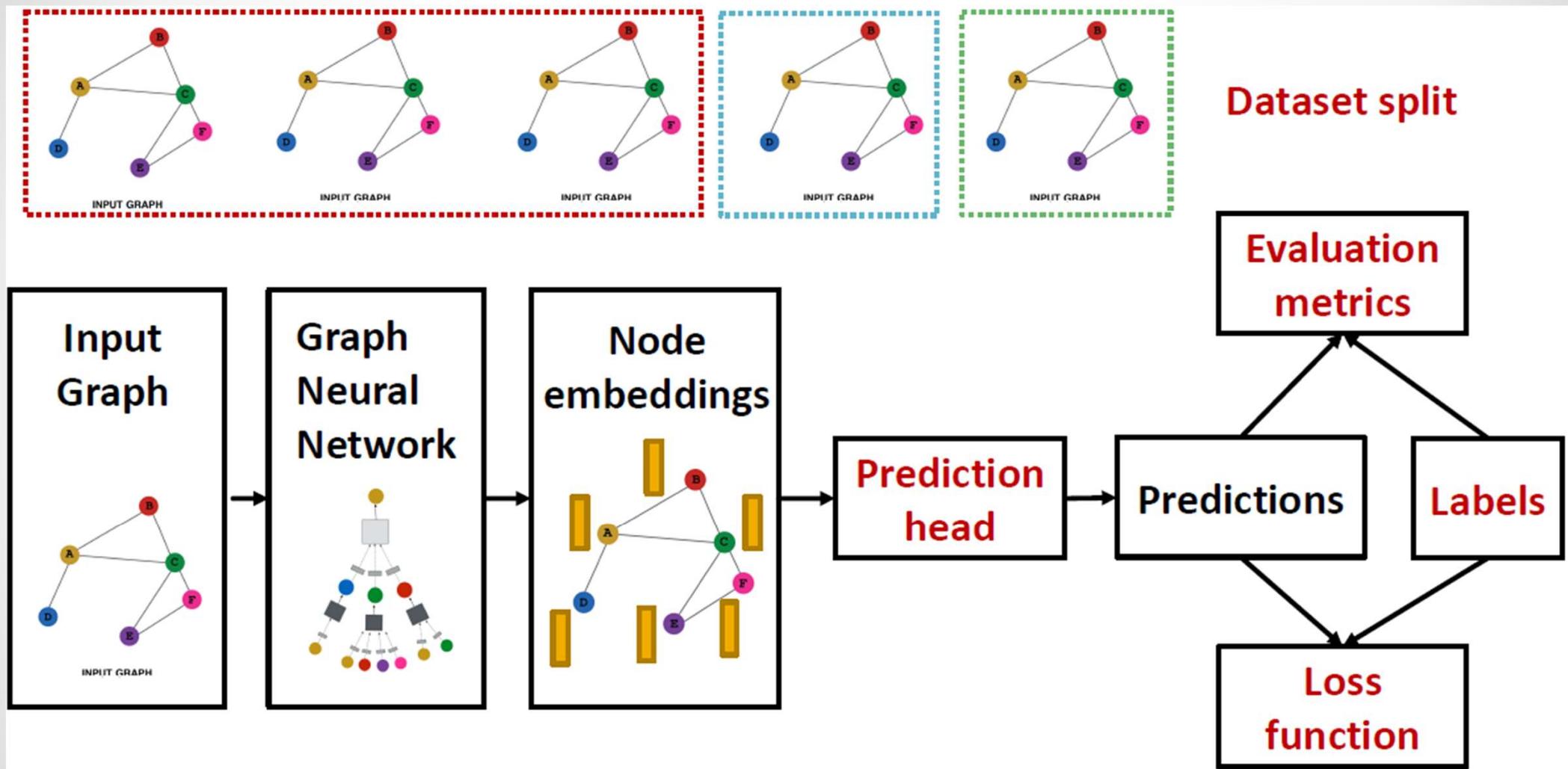


(2) At validation time:  
Use **training message edges & training supervision edges** to predict **validation edges**



(3) At test time:  
Use **training message edges & training supervision edges & validation edges** to predict **test edges**

# GNN Training Pipeline



# Summary

- We introduce **a general perspective for GNNs**
  - **GNN Layer:**
    - Transformation + Aggregation
    - Classic GNN layers: GCN, GraphSAGE, GAT
  - **Layer connectivity:**
    - The over-smoothing problem
    - Solution: skip connections
  - **Graph Augmentation:**
    - Feature augmentation
    - Structure augmentation
  - **Learning Objectives**
    - The full training pipeline of a GNN