

GNN Training & Evaluation

소프트웨어 끈대 강의

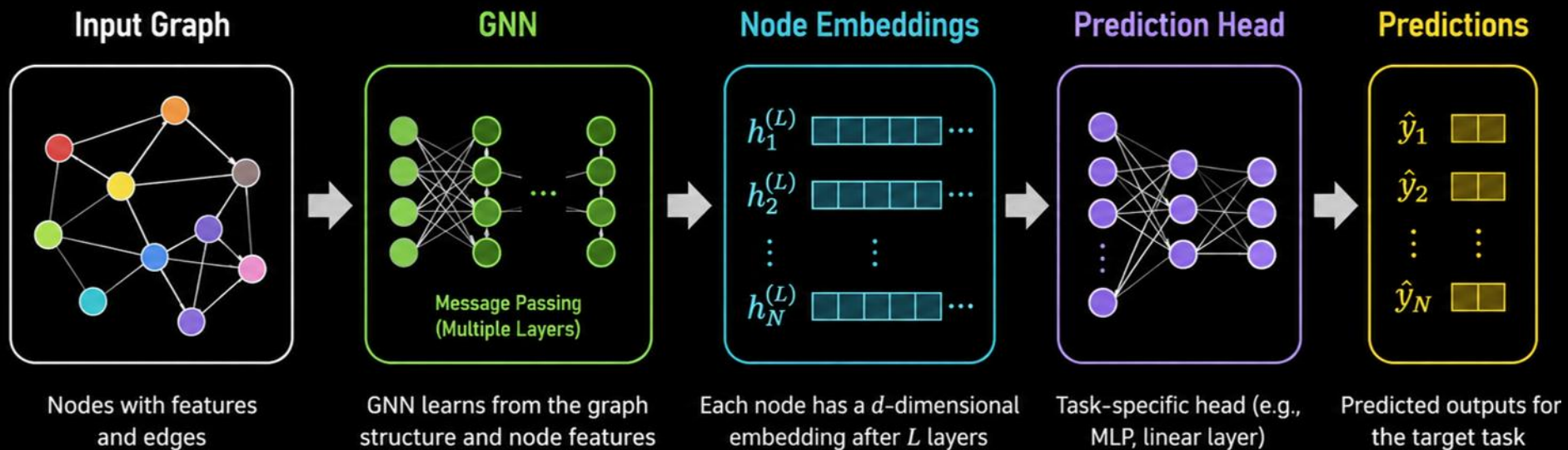
노기섭 교수

(kafa46@gmail.com)

Contents

- **Prediction Heads for GNNs**
 - Node-level Prediction
 - Edge-level Prediction
 - Graph-level Prediction
- **Training & Evaluation**
 - Loss Functions
 - Evaluation Metrics
 - ROC-AUC
- **Dataset Splitting for GNNs**
 - Transductive vs. Inductive Learning
 - Node Classification Split
 - Graph Classification Split
 - Link Prediction Split

From GNN Embeddings to Predictions



We have learned how to generate node embeddings.

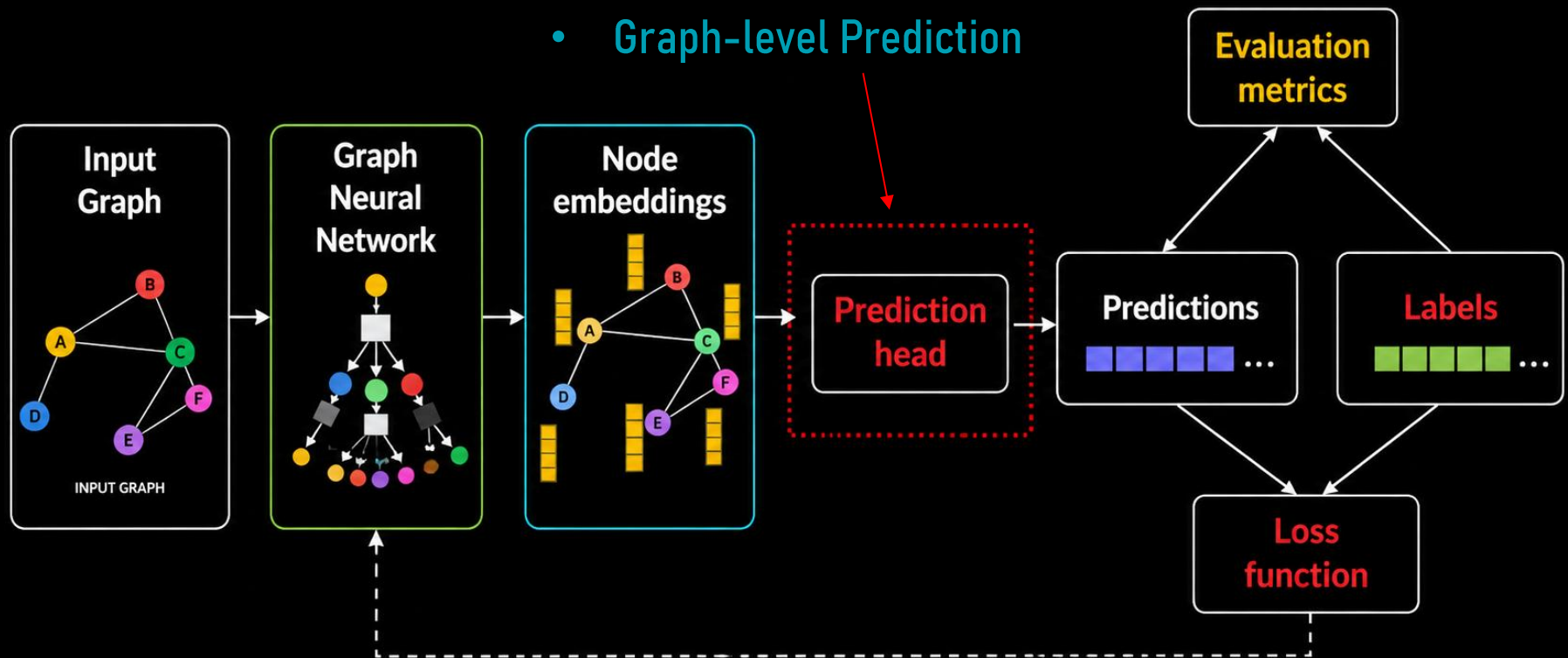
What do we do with them?

Prediction Heads for GNNs

Prediction Heads for GNNs

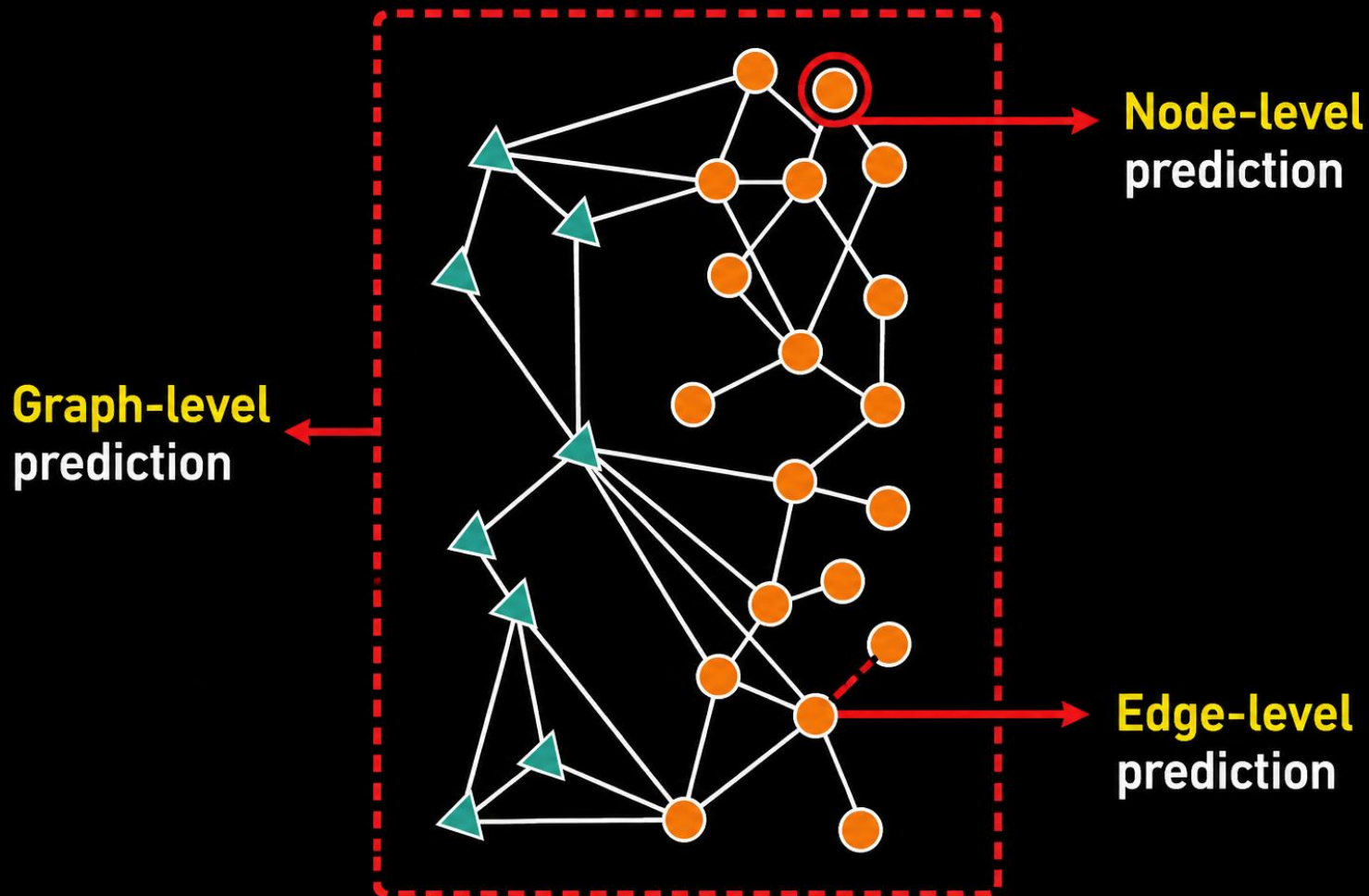
Task Types of Prediction Heads:

- Node-level Prediction
- Edge-level Prediction
- Graph-level Prediction



End-to-End Training with Backpropagation

Details in Prediction Tasks



Node-Level Prediction

Node-level Prediction:

Directly make prediction using node embeddings

GNN Output: $h_v^L \in \mathbb{R}^d$

Each node has a d -dimensional embedding.

Node Prediction Head: $\hat{y}_v = \mathbf{Head}_{\text{node}}(h_v^L) = W^H h_v^L$

Decision Mapping: $\mathbb{R}^d \rightarrow \mathbb{R}^k$, $W^H \in \mathbb{R}^{k \times d}$

- Classification: Predict one of k classes.
- Regression: Predict k target values.

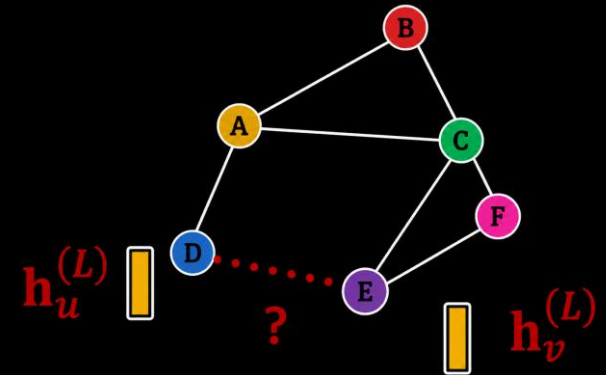
Edge-Level Prediction

Edge-level Prediction:

Make predictions for a pair of nodes using their node embeddings.

GNN Output: $h_u^L, h_v^L \in \mathbb{R}^d$

Each node has a d-dimensional embedding.



Edge Prediction Head: $\hat{y}_v = \mathbf{Head}_{\text{edge}}(h_u^L, h_v^L) = W^H h_v^L$

- Concatenation: $[h_u^L || h_v^L]$
- Dot Product: $h_u^{L^T} \cdot h_v^L$
- MLP: learn a nonlinear relation between two nodes

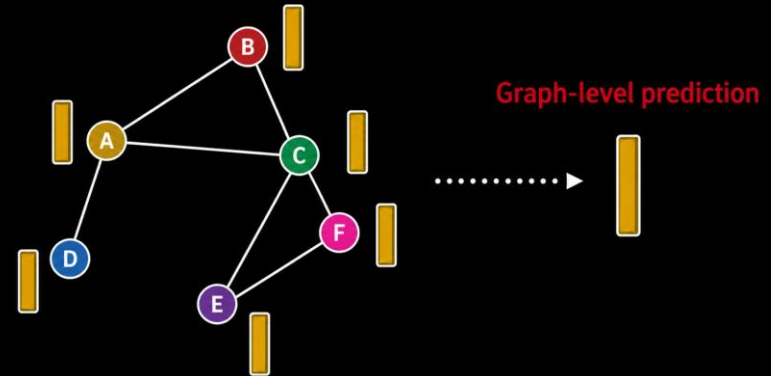
Decision Mapping:

- Link Prediction: Does an edge exist between u and v ?
- Edge Classification: What type of relation connects u and v ?

Graph-Level Prediction

Graph-level Prediction:

Make prediction using all the node embeddings in our graph.



GNN Output: $h_u^L \in \mathbb{R}^d, \forall v \in G$

Each node has a d-dimensional embedding.

Prediction Head: $\hat{y}_G = \mathbf{Head}_{\text{graph}}(\{h_u^L \in \mathbb{R}^d, \forall v \in G\}) = W^H h_v^L$

Decision Mapping:

- Molecule $\rightarrow$ Toxic / Non-toxic
- Protein $\rightarrow$ Protein function
- Program graph $\rightarrow$ Vulnerable / Safe
- Social network $\rightarrow$ Network type

⋮

- Predicts a label or property for the entire graph
- Combines all node embeddings into a single graph representation

Graph-Level Prediction: Options

Prediction Head:

$$\hat{y}_G = \mathbf{Head}_{\text{graph}}(\{h_u^L \in \mathbb{R}^d, \forall v \in G\}) = W^H h_v^L$$

Works well in
small graphs!



What if
large graphs?

Global Mean Pooling

$$\hat{y}_G = \mathbf{Mean}(\{h_u^L \in \mathbb{R}^d, \forall v \in G\}) = W^H h_v^L$$

Global Max Pooling

$$\hat{y}_G = \mathbf{Max}(\{h_u^L \in \mathbb{R}^d, \forall v \in G\}) = W^H h_v^L$$

Global Sum Pooling

$$\hat{y}_G = \mathbf{Sum}(\{h_u^L \in \mathbb{R}^d, \forall v \in G\}) = W^H h_v^L$$

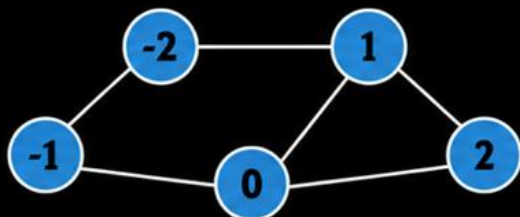
Graph-Level Prediction: Global Polling - Issue

Issue: Global pooling over a (large) graph will lose information

- Clearly G_1 and G_2 have very different node embeddings

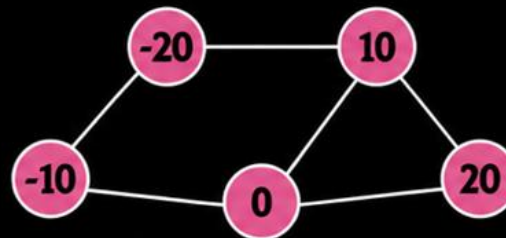
→ Their structures should be different

G_1



Node embeddings for G_1 : $\{-1, -2, 0, 1, 2\}$

G_2



Node embeddings for G_2 : $\{-10, -20, 0, 10, 20\}$

If we do **GLOBAL SUM POOLING**:

Let's sum
all node
embeddings!



Prediction for G_1 :

$$\begin{aligned}\hat{y}_G &= \text{Sum}(\{-1, -2, 0, 1, 2\}) \\ &= -1 - 2 + 0 + 1 + 2 \\ &= 0\end{aligned}$$

Prediction for G_2 :

$$\begin{aligned}\hat{y}_G &= \text{Sum}(\{-10, -20, 0, 10, 20\}) \\ &= -10 - 20 + 0 + 10 + 20 \\ &= 0\end{aligned}$$

Oh? The
sum is also
zero?!



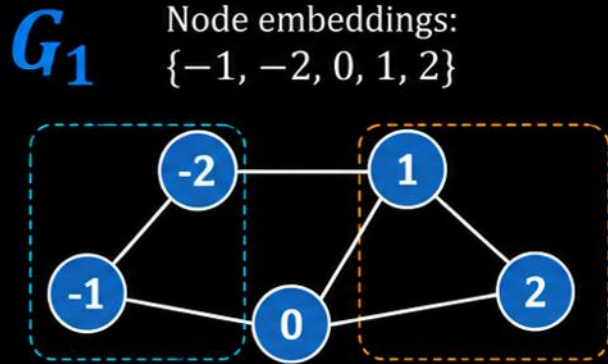
We cannot differentiate G_1 and G_2 !

OOPS!

Graph-Level Prediction: Global Polling - Solution 1

Solution 1: aggregate all the node embeddings hierarchically

Grouping + ReLU



Round 1: separate aggregation

First 2 nodes $(-1, -2)$
Sum $= (-1) + (-2) = -3$

↓
 $\text{ReLU}(-3) = 0$
 $\Rightarrow \hat{y}_a = 0$

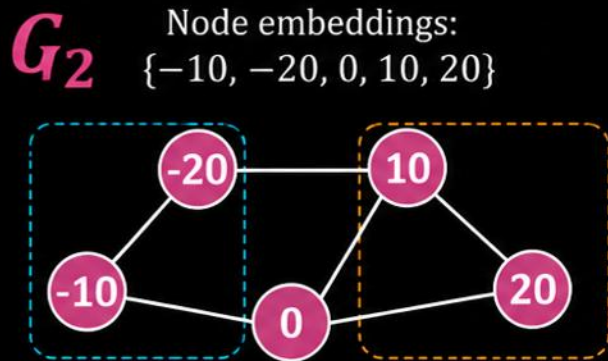
Last 3 nodes $(0, 1, 2)$
Sum $= 0 + 1 + 2 = 3$

↓
 $\text{ReLU}(3) = 3$
 $\Rightarrow \hat{y}_b = 3$

Round 2: final aggregation

Sum of $\hat{y}_a$ and $\hat{y}_b$
Sum $= 0 + 3 = 3$

↓
 $\text{ReLU}(3) = 3$
 $\hat{y}_G = 3$



Round 1: separate aggregation

First 2 nodes $(-10, -20)$
Sum $= (-10) + (-20) = -30$

↓
 $\text{ReLU}(-30) = 0$
 $\Rightarrow \hat{y}_a = 0$

Last 3 nodes $(0, 10, 20)$
Sum $= 0 + 10 + 20 = 30$

↓
 $\text{ReLU}(30) = 30$
 $\Rightarrow \hat{y}_b = 30$

Round 2: final aggregation

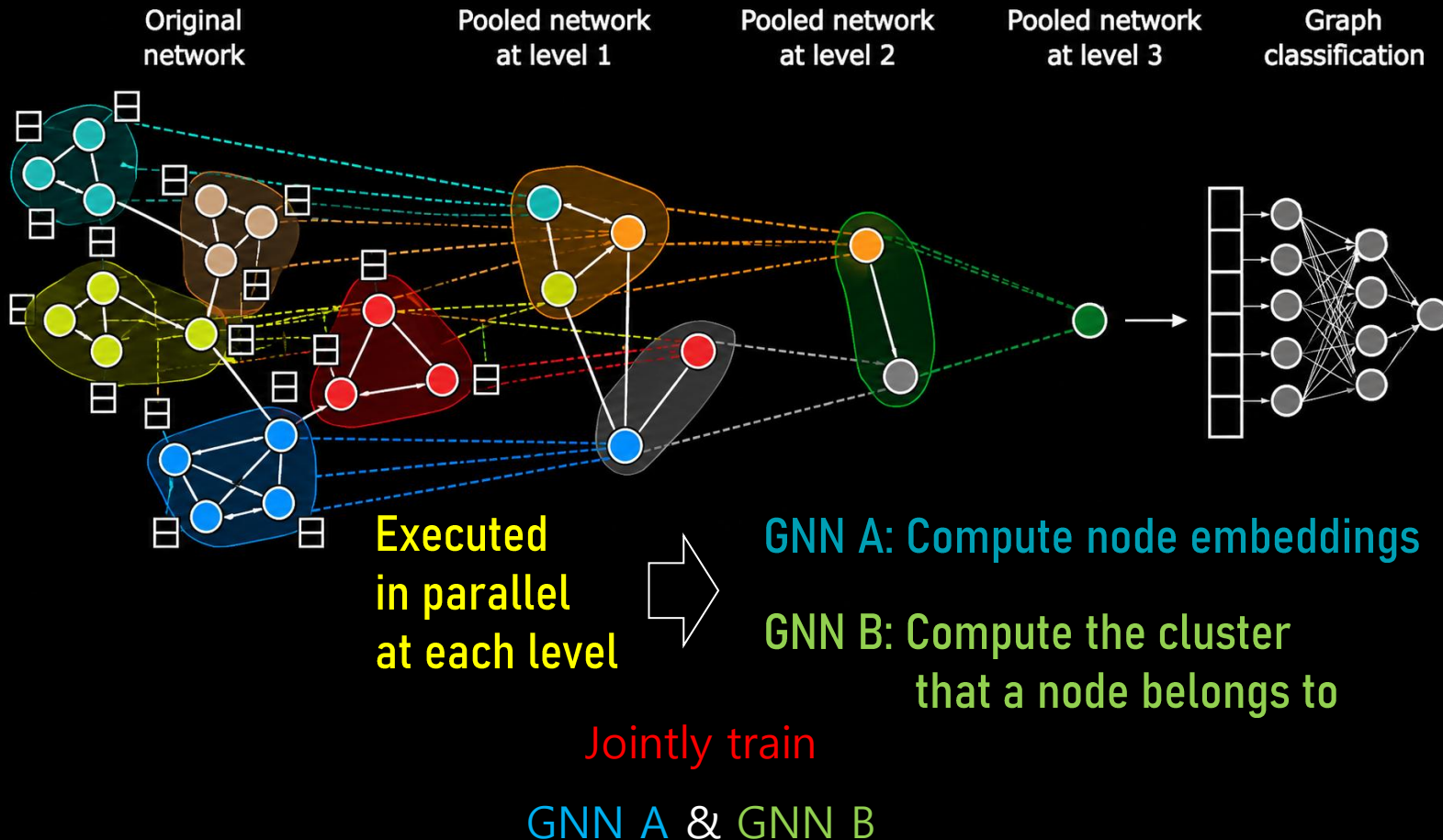
Sum of $\hat{y}_a$ and $\hat{y}_b$
Sum $= 0 + 30 = 30$

↓
 $\text{ReLU}(30) = 30$
 $\hat{y}_G = 30$

Graph-Level Prediction: Global Polling - Solution 2

Solution 1: DiffPool [NeurIPS 2028]

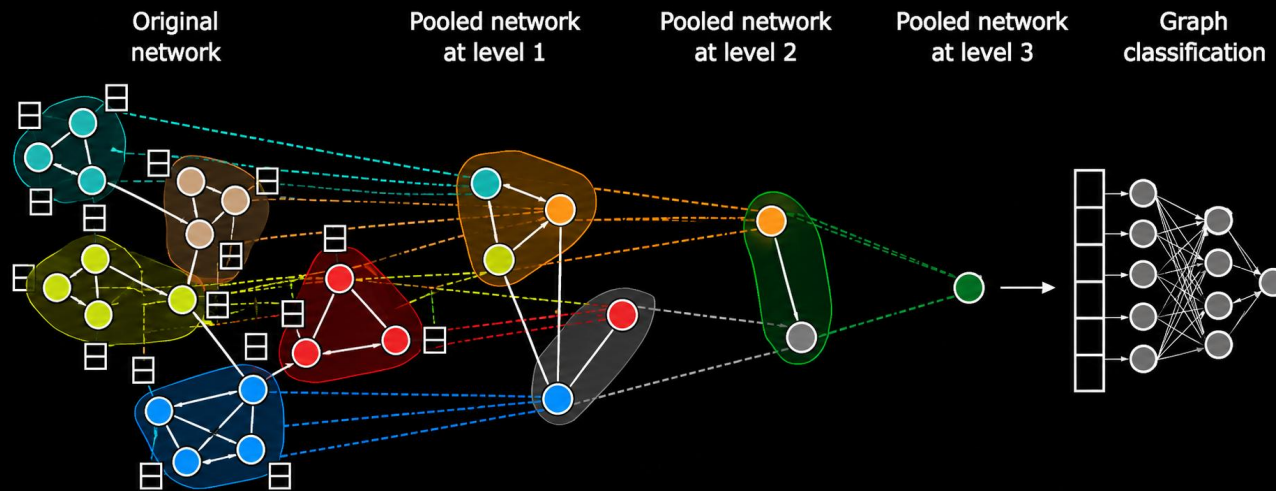
Leverage two independent GNNs at each level



Graph-Level Prediction: Global Polling - Solution 2

Solution 1: DiffPool [NeurIPS 2028]

Leverage two independent GNNs at each level



For each Pooling layer

- Use clustering assignments from **GNN B** to aggregate node embeddings generated by **GNN A**
- Create a single new node for each cluster, maintaining edges between clusters to generate a new pooled network

Training & Evaluation

From Predictions to Learning

Predictions Are Not Enough

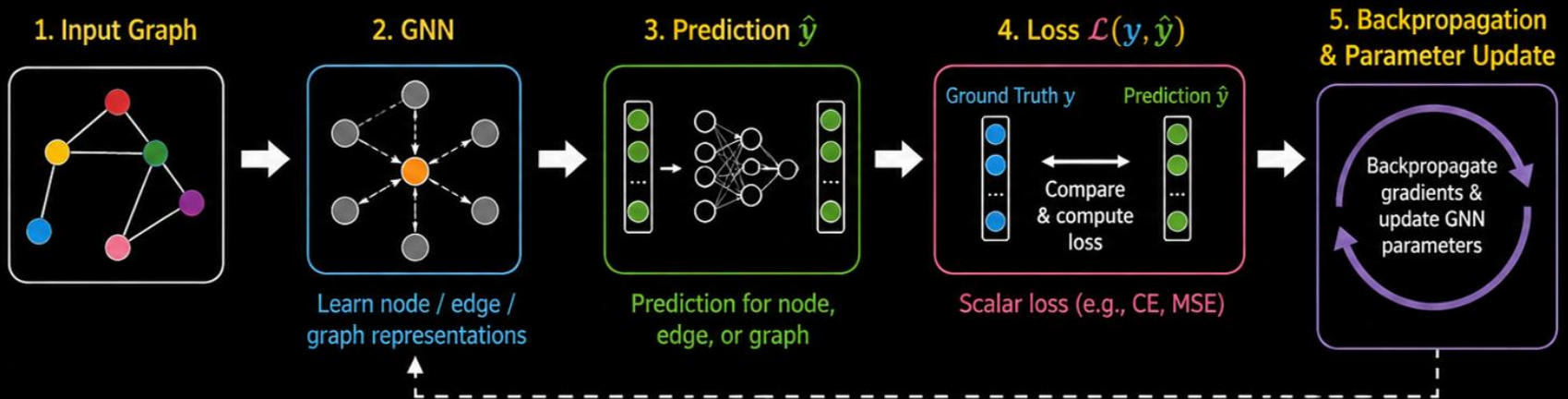
- A GNN produces a prediction $\hat{y}$ for a node, edge, or graph.
- During training, we also have the ground-truth target y .
- Measure how different prediction from target
- Loss function converts this difference into a scalar value.

Node-level: prediction $\hat{y}_v^i$, label y_v^i

Edge-level: prediction $\hat{y}_{uv}^i$, label y_{uv}^i

Graph-level: prediction $\hat{y}_G^i$, label y_G^i

Loss Functions for GNNs



GNN Task	Typical Output	Common Loss
Node Classification	Class of each node	Cross-Entropy Loss
Edge / Link Prediction	Edge existence probability	Binary Cross-Entropy
Graph Classification	Class of the whole graph	Cross-Entropy Loss
Node / Graph Regression	Continuous value	MSE / MAE

Cross-Entropy Loss

Cross-Entropy for Classification

- Used for multi-class classification
- The prediction head produces a probability distribution over C classes

$$\hat{y}_v = [\hat{y}_{v1}, \hat{y}_{v2}, \dots, \hat{y}_{vC}]$$

- The ground-truth label y_i is the representation as a one-hot vector

Loss of a Node:

$$\mathcal{L}_v = - \sum_{c=1}^C y_{vc} \log \hat{y}_{vc}$$

- y_{vc} : ground-truth label for class c
- $\hat{y}_{vc}$: predicted probability for class c

Total Train Loss: $\mathcal{L}_v = \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{c=1}^C \mathcal{L}_v$



Good Prediction

Ground Truth y_v

$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$
(Class 2)

Prediction $\hat{y}_v$

0.05	0.90	0.05
------	------	------

Loss: $\mathcal{L}_v = -\log(0.90) \approx 0.105$



Bad Prediction

Ground Truth y_v

$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$
(Class 2)

Prediction $\hat{y}_v$

0.80	0.10	0.10
------	------	------

Loss: $\mathcal{L}_v = -\log(0.10) \approx 2.303$

Mean Squared Error (MSE) for Regression

Regression predicts a continuous value.

The prediction error for one sample is:

$$e_i = y_i - \hat{y}_i$$

Mean Squared Error (MSE):

Measures the average squared prediction error

$$\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Small MSE is better
prediction performance

Training Goal: Minimize MSE $\rightarrow \hat{y}_i \approx y_i$

How Do We Evaluate a GNN?

Task	Output	Common Metrics
Classification	Discrete class	Accuracy, Precision, Recall, F1-score
Regression	Continuous value	MSE, MAE, RMSE
Ranking / Link Prediction	Ranked candidates	MRR, Hits@K, AUC

Classification

- Accuracy: fraction of correctly predicted samples
- Precision / Recall / F1: useful for imbalanced classes

Regression

- Measures how far predictions are from true values
- MSE gives larger penalties to large errors

Link Prediction / Ranking

- Evaluate whether the model assigns higher scores to correct links
- Common metrics: MRR, Hits@K, AUC

Accuracy: A Simple Evaluation Metric

What is Accuracy?

- Measures the fraction of correctly predicted samples
- Simple and intuitive metric for classification tasks
- Used for node-level or graph-level classification

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Simple Example

10 nodes are classified:



✓ Correct predictions: 8

✗ Incorrect predictions: 2

$$\text{Accuracy} = 8 / 10 = 0.8 = 80\%$$

When Does Accuracy Work Well?

- Classes are relatively balanced
- Correct predictions are equally important across classes



Limitation: Can Be Misleading for Imbalanced Data

Example: 95 normal nodes + 5 anomalous nodes



Predict All as
Normal



$$\text{Accuracy} = 95 / 100 = 95\% \text{ (But detects 0 anomalies!)}$$

ROC Curve: TPR vs. FPR

ROC Curve:

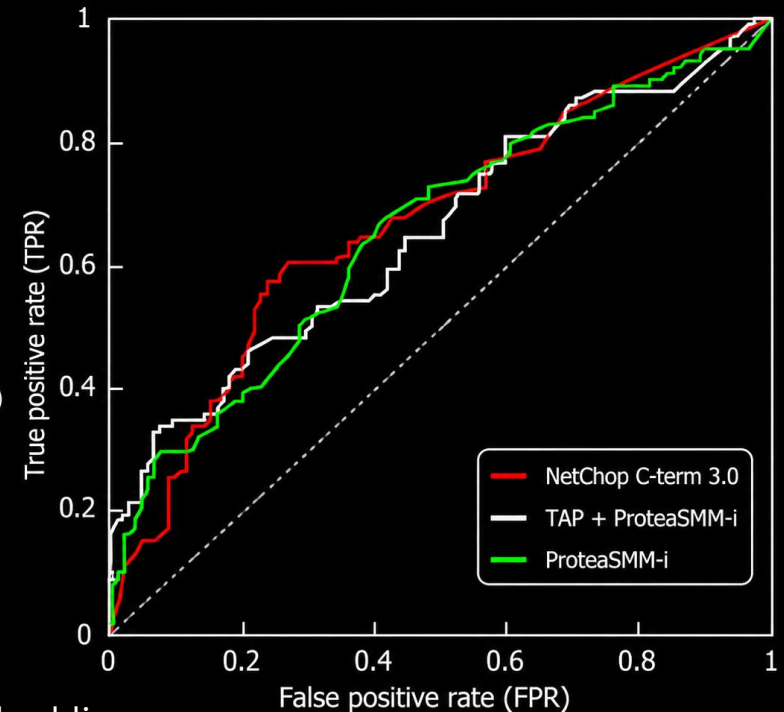
Captures the tradeoff in TPR and FPR as the classification threshold is varied for a binary classifier.

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

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

$$FPR = \frac{FP}{FP + FN}$$

TPR
(True Positive Rate)



FPR
(False Positive Rate)

Note: the dashed line represents performance of a random classifier

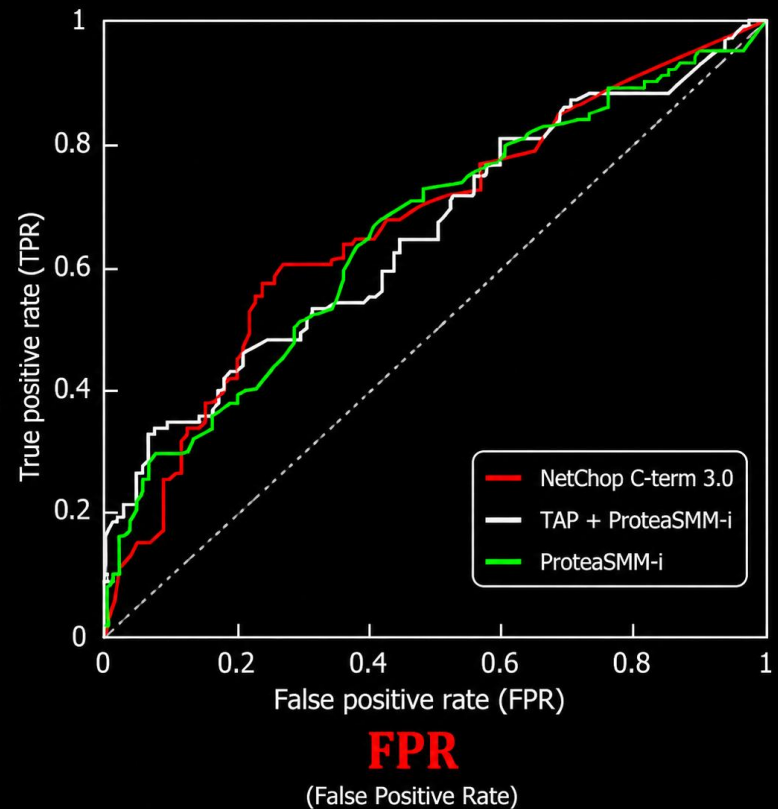
ROC-AUC: Threshold-Independent Evaluation

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

TPR
(True Positive Rate)



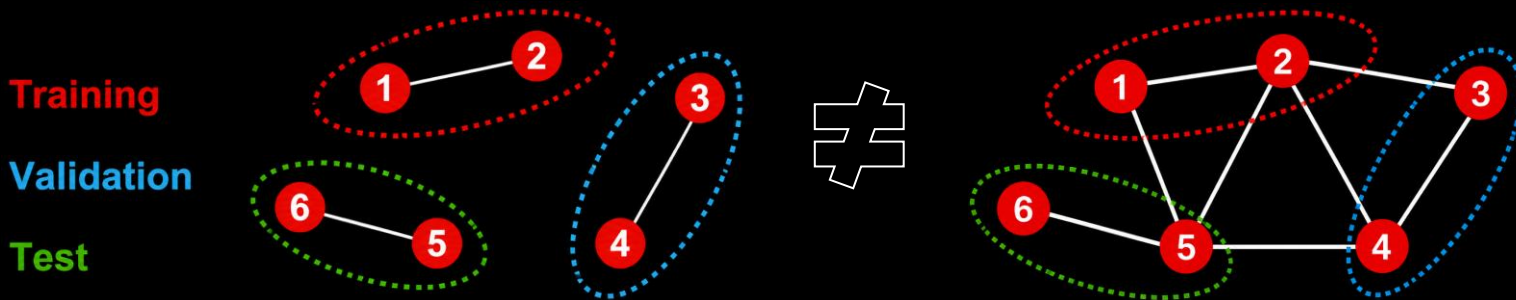
Note: the dashed line represents performance of a random classifier

Dataset Splitting for GNNs

Why Is Dataset Splitting Different for Graphs?

Traditional Machine Learning

- Samples are usually treated as **independent and identically distributed (IID)**.
- Train / validation / test sets **can be split independently**.
- Splitting one sample does **not affect other samples**.



Graph Machine Learning

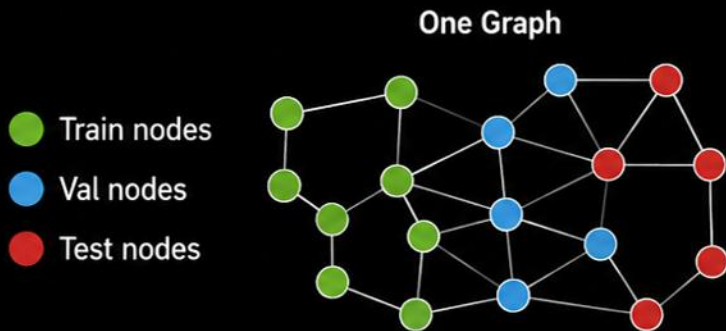
- Nodes **are connected through edges** and are **not independent**.
- A node can **receive information from nodes** in different data splits.
- Graph **structure may be shared** across train / validation / test sets.

Therefore, **what is visible during message passing must be carefully controlled.**

Transductive vs. Inductive Learning

Transductive Learning

Learn and predict on the **same graph**.

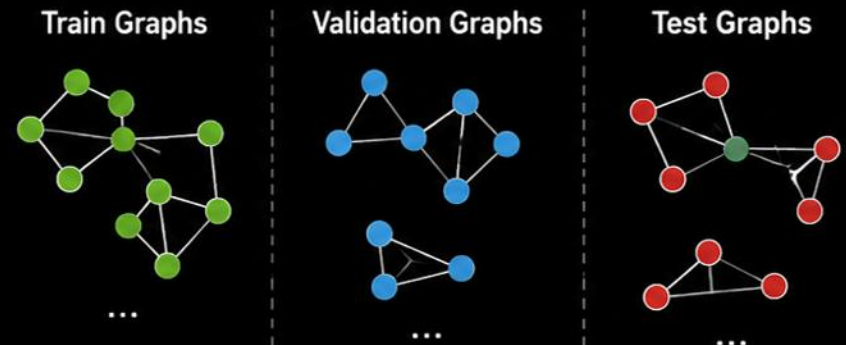


The model has access to the **entire graph structure** (including test nodes) during message passing.

- Typical in citation networks, social networks
- Goal: predict labels for (some) nodes in the **same graph**

Inductive Learning

Generalize to **unseen graphs**.



The model is trained on some graphs and evaluated on different, **unseen graphs**.

- Common in molecular graphs, recommendation, or any setting with new graphs over time
- Goal: generalize to entirely **new graphs**

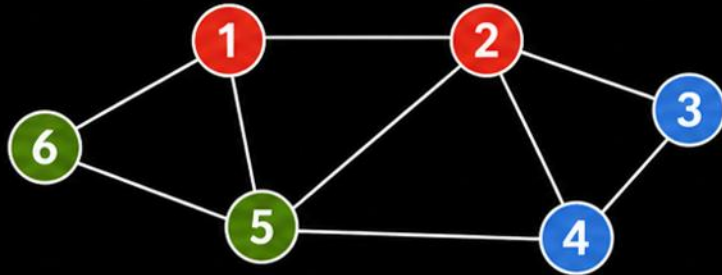


Key Takeaway: Transductive assumes test data comes from the **same graph**.
Inductive assumes test data comes from **different graphs**.

Transductive Node Classification

In the transductive setting, we **split node labels** but **observe the entire graph**.

One Graph (all nodes and edges are observed)



● Training nodes (labels used for training)

● Validation nodes (labels used for validation)

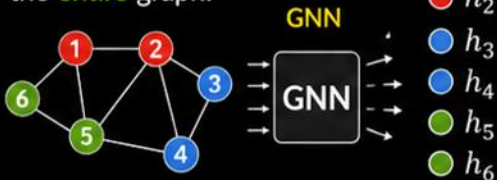
● Test nodes (labels used for testing)



Key idea: We split the node labels, not the graph.

1) Training

Compute embeddings using the **entire** graph.

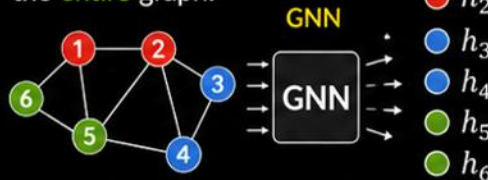


Compute loss using only **training labels**.

$$\mathcal{L}_{train} = \mathcal{L}(h_1, y_1) + \mathcal{L}(h_2, y_2)$$

2) Validation

Compute embeddings using the **entire** graph.

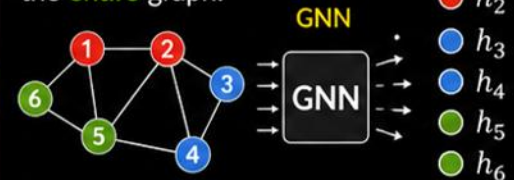


Evaluate using **validation labels**.

Evaluate on $\{y_3, y_4\}$

3) Test

Compute embeddings using the **entire** graph.



Evaluate using **test labels**.

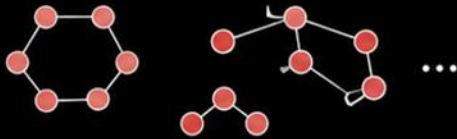
Evaluate on $\{y_5, y_6\}$

The entire graph is visible, but only training labels are used for learning.

Inductive Node Classification

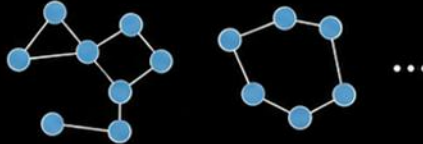
In the inductive setting, we **generalize** to **unseen graphs**.

Training Graphs (seen during training)



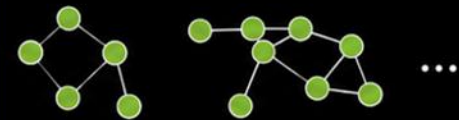
- Node labels are **available**.
- Model learns from these graphs.

Validation Graphs (unseen during training)

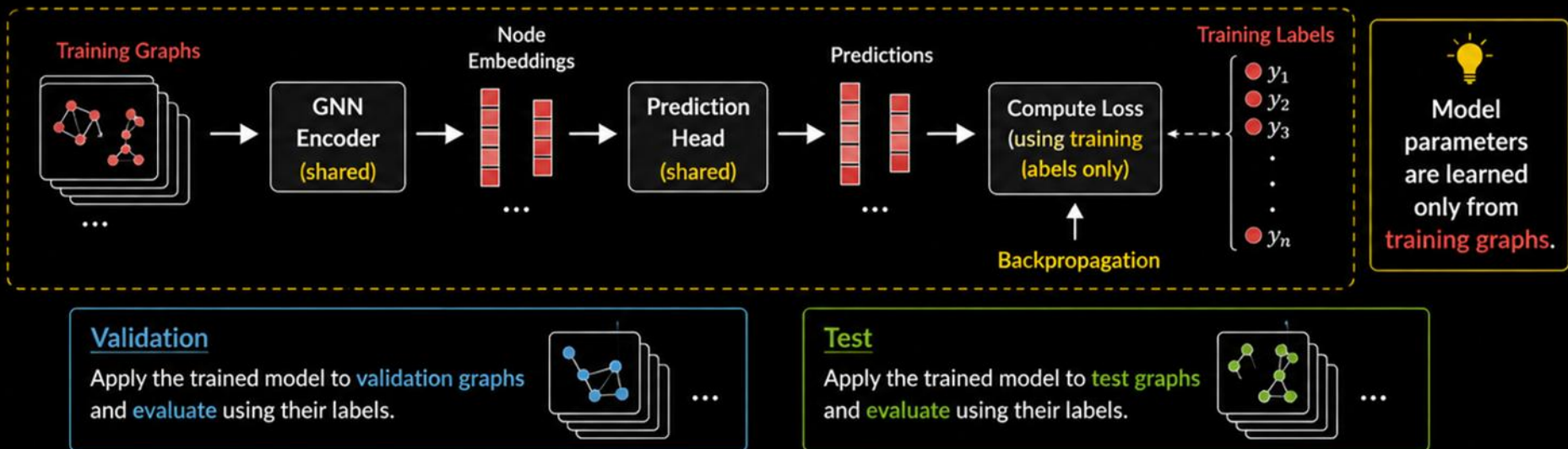


- Node labels are **available**.
- Used only for model selection.

Test Graphs (unseen during training)



- Node labels are **available**.
- Used only for final evaluation.

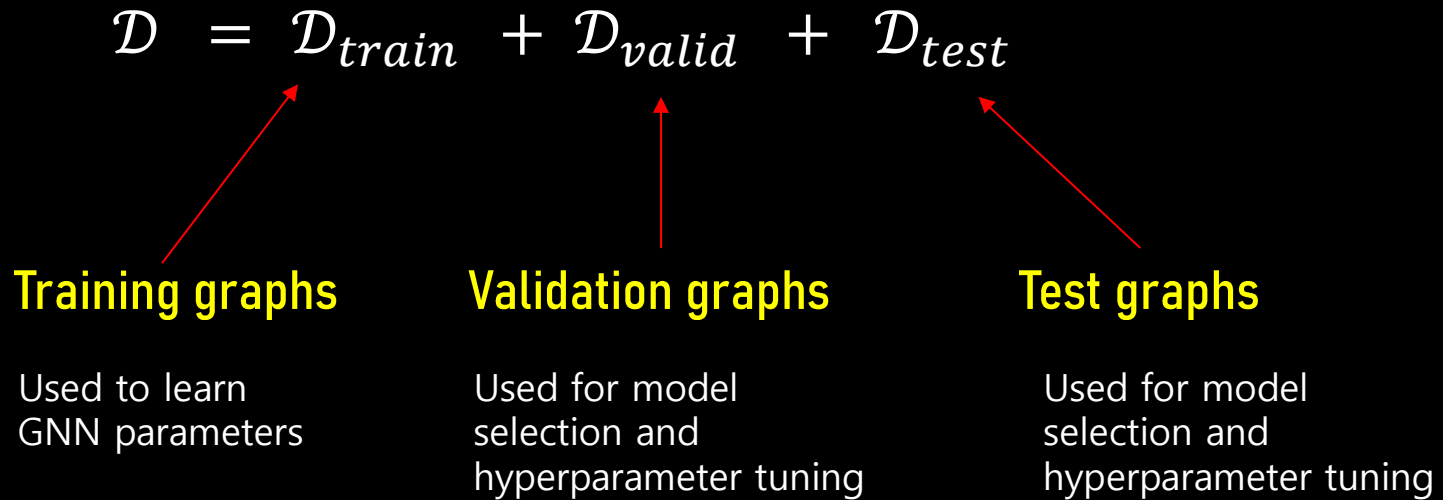


Inductive node classification trains on **seen graphs** & generalizes to **unseen graphs**

Dataset Split for Graph Classification (1/2)

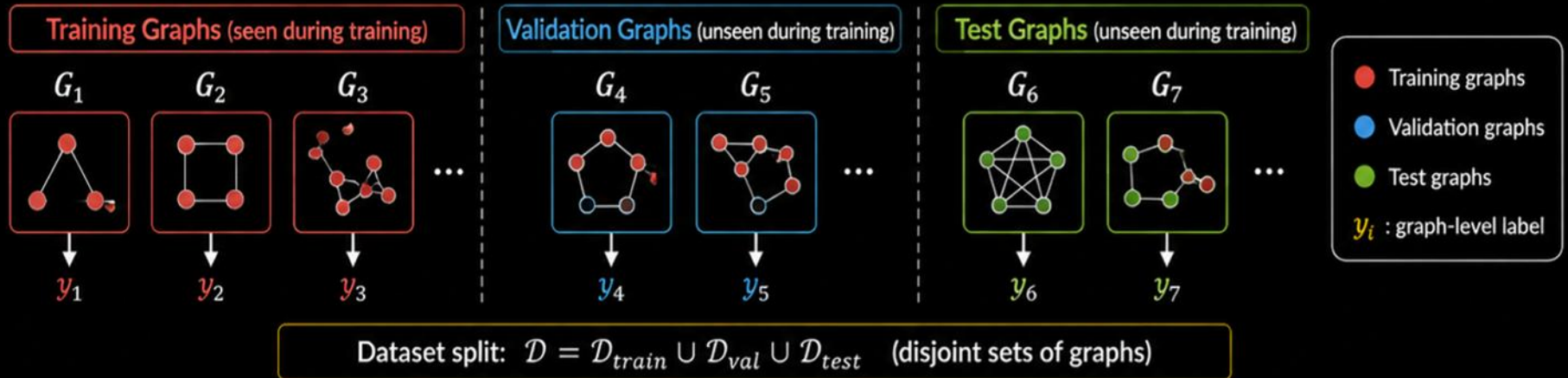
Graphs Are Individual Samples

- Each graph G_i is treated as one independent sample.
- Each graph has its own graph-level label y_i .
- We split the graphs themselves into training, validation, and test sets.

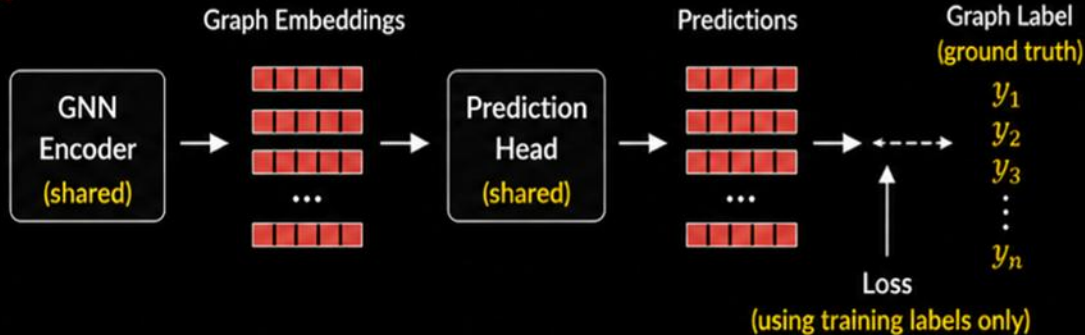


Dataset Split for Graph Classification (2/2)

In graph classification, each graph is an **independent sample** with a **graph-level label**.



Input: Batch of Graphs



Key Points

- Each graph is an **independent** sample.
- We split **entire graphs**, not nodes within a graph.
- Model parameters are learned only from **training graphs**.
- The model must generalize to **unseen graphs**.

Validation

Apply the trained model to **validation graphs** (unseen during training) and evaluate using their labels.



Test

Apply the trained model to **test graphs** (unseen during training) and evaluate using their labels.

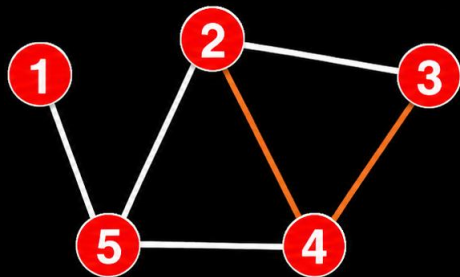


Dataset Split for Link Prediction

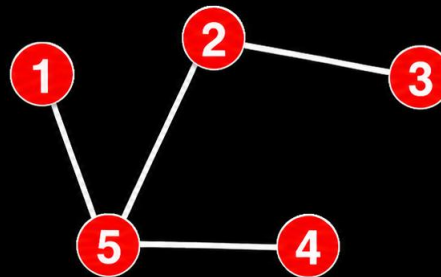
Goal of link prediction: predict missing edges

Setting up link prediction is tricky:

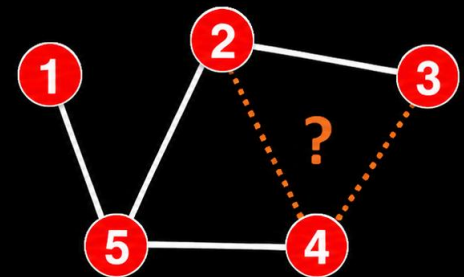
- Link prediction is an unsupervised / self-supervised task.
- We need to create the labels and dataset splits on our own
- Concretely, we need to hide some edges from the GNN and let the GNN predict if the edges exist



Original graph



Input graph to GNN



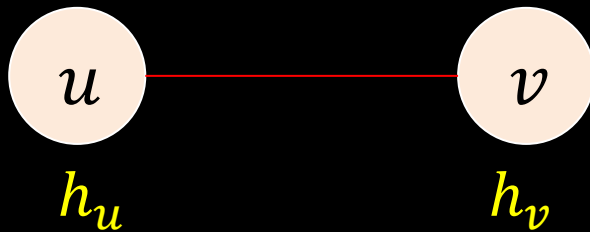
Predictions made by GNN

Message Edges vs. Supervision Edges

Link Prediction is Different!

- The goal is to predict whether an edge exists between two nodes.
- Unlike node or graph classification, edges play two different roles:
 - They define the graph structure for message passing
 - They can also be the targets we want to predict

Why Cannot We Simply Split Edges?



But already the edge is used
during message passing



$$h_u^l = \text{Agg}(h_u^h, \{h_v^{l-1} : v \in N(u)\})$$

이미 답을 보여주고 문제를 풀라고 하는 상황

Message Edges vs. Supervision Edges

Edge Type

Role

Message Edge

- Used for GNN message passing

Supervision Edge

- GNN predict this edge
- Use for computing objectives (Prediction target / loss computing)

1) Message Edges ($E_{message}$)

- Used as input to the GNN
- Define the graph structure for message passing
- Used to compute node embeddings

$$E_{message} \rightarrow \text{GNN} \rightarrow h_u, h_v$$

2) Supervision Edges ($E_{supervision}$)

- Used as prediction targets
- NOT used for message passing
- Used to compute training loss

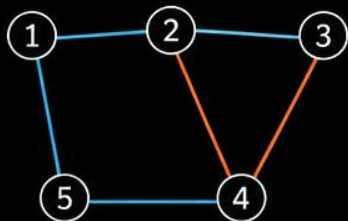
$$(h_u, h_v) \rightarrow \text{Link Predictor} \rightarrow \hat{y}_{uv}$$

- Message edges ($E_{message}$) used for GNN
- Supervision edges ($E_{supervision}$) prediction targets
- Nodes

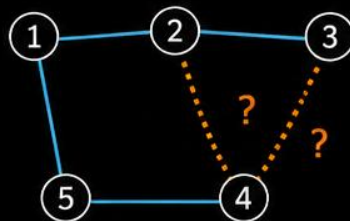


Do not use an edge as a message edge when that same edge is the prediction target.

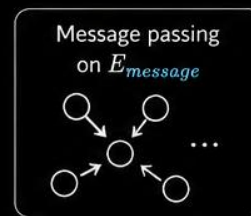
Original Graph



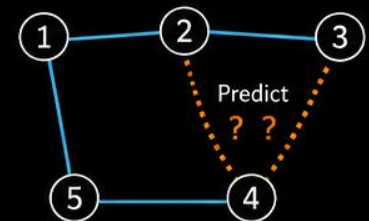
Input Graph to GNN (message edges only)



GNN



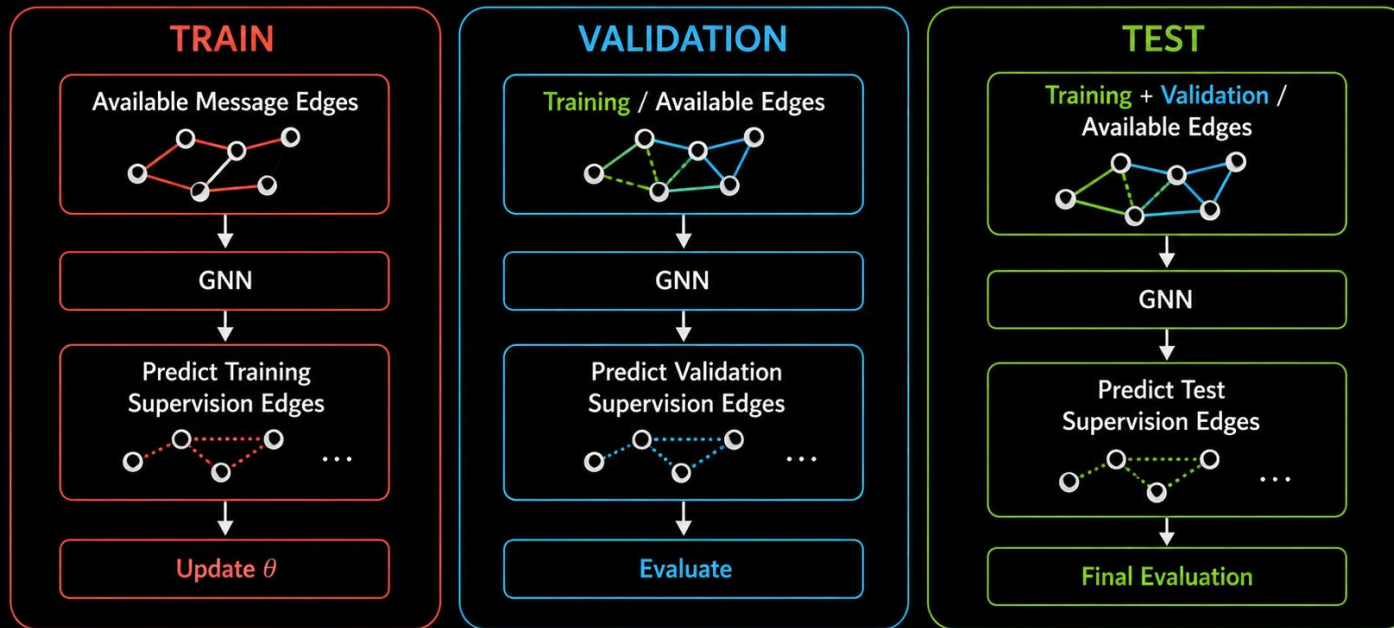
Predictions made by GNN



Transductive Link Prediction Split

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



Key Takeaway: At each stage, predict held-out supervision edges using only the message edges available at that stage.



Never reveal the current target edges during message passing.

Information Leakage in Link Prediction

Information Leakage?

- Target edges **must not be visible** during message passing
- Otherwise, GNN can **indirectly see the answer** it is supposed to predict
- This leads to **overly optimistic** validation/test performance

Key Principle

Separate **Message-Passing Edges** from **Supervision Edges**

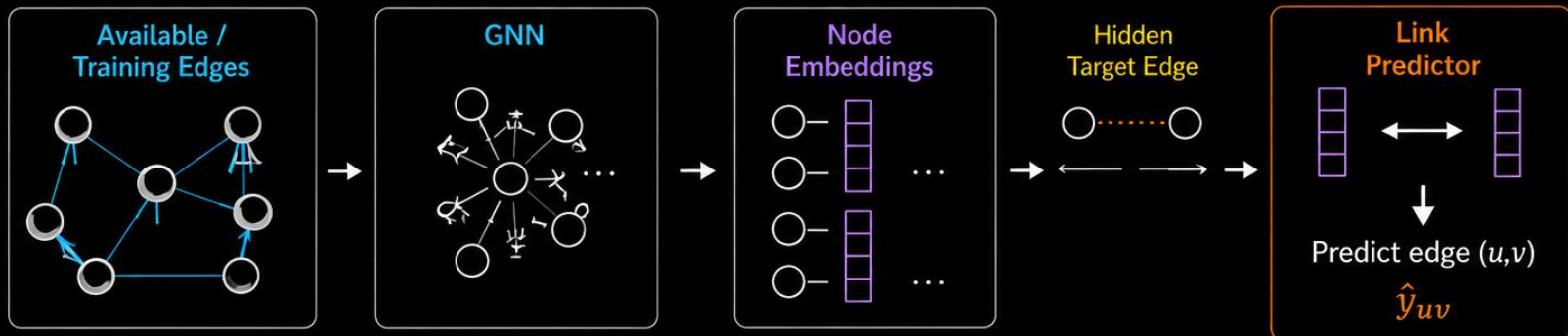
1 Message-Passing Edges

- Used by the GNN to compute node embeddings
- Define what structural information the model can access

2 Supervision Edges

- Used only to compute the prediction loss or evaluation metric
- Must remain **hidden** from the GNN when they are prediction targets

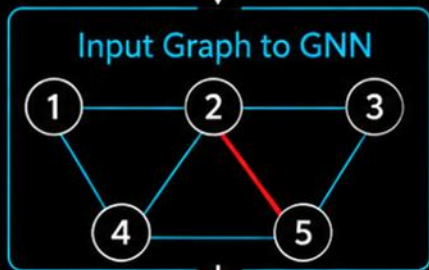
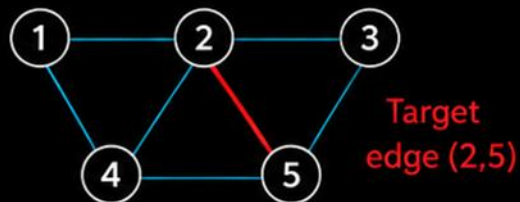
Correct Link Prediction Setup



Example: Information Leakage

✗ **WRONG**

Target edge is present in the input graph.



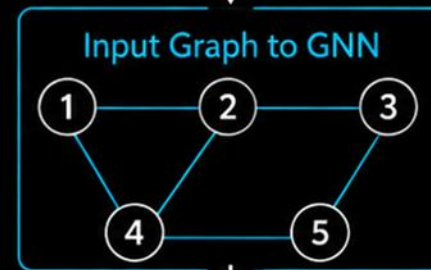
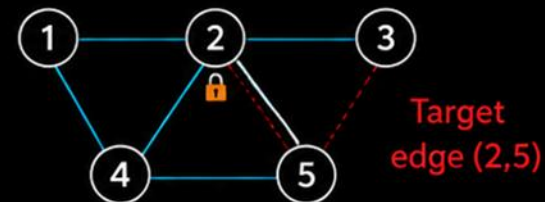
Predict edge (2,5)



GNN **already sees** the target edge during message passing!

✓ **CORRECT**

Target edge is hidden from the input graph.



Predict edge (2,5)



GNN must **infer the missing** edge from the available structure.

Summary

Summary (1/2)

Graph data splitting is different from ordinary IID data splitting.

- Nodes and edges are structurally connected.
- Information can flow across dataset splits.

Transductive and inductive learning use different split strategies.

- Transductive: the same graph is observed across training, validation, and test.
- Inductive: the model must generalize to unseen nodes or graphs.

The split strategy depends on the prediction task.

- Node classification: split node labels or graphs.
- Graph classification: split entire graphs.
- Link prediction: split edges carefully.

Summary (2/2)

Link prediction requires separating edge roles.

- Message edges: used for GNN message passing.
- Supervision edges: used as prediction targets and for loss computation.

Target edges must remain hidden during prediction.

- Otherwise, the GNN can see the answer through message passing.
- This causes information leakage and overly optimistic evaluation.



수고하셨습니다 ..^^..