Data Mining
Graph Mining

https://data-mining.github.io/winter-2026/

CS 453/553 – Winter 2026
Yu Wang, Ph.D.
Assistant Professor
Computer Science
University of Oregon

Fun Facts – Safety of Embodied AI

In general any ML problem – Maintain Four Pieces

Data	Model	Loss	Optimization

$\{(x^i, y^i)\}_{i=1}^{N}$

$\{(x^i, y^i)\}_{i=1}^{N_{\text{train}}}$

$\{(x^i, y^i)\}_{i=1}^{N_{val}}$

$\{(x^i, y^i)\}_{i=1}^{N_{test}}$

$y = f_\theta(x)$

f: f formats decide the potential

θ: decides whether you can achieve that potential

Metric between predicted and ground-truth

In general any ML problem – Maintain Four Pieces

Data | **Model** | **Loss** | **Optimization**

$$\{(x^i, y^i)\}_{i=1}^{N}$$

$$\{(x^i, y^i)\}_{i=1}^{N\text{train}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{val}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{test}}$$

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_excel("HousePricePrediction.xlsx")

# Printing first 5 records of the dataset
print(dataset.head(5))
```

```
   Id  MSSubClass MSZoning  LotArea LotConfig BldgType  OverallCond  \
0   0          60       RL     8450    Inside     1Fam            5
1   1          20       RL     9600       FR2     1Fam            8
2   2          60       RL    11250    Inside     1Fam            5
3   3          70       RL     9550    Corner     1Fam            5
4   4          60       RL    14260       FR2     1Fam            5

   YearBuilt  YearRemodAdd Exterior1st  BsmtFinSF2  TotalBsmtSF  SalePrice
0       2003          2003     VinylSd         0.0        856.0   208500.0
1       1976          1976     MetalSd         0.0       1262.0   181500.0
2       2001          2002     VinylSd         0.0        920.0   223500.0
3       1915          1970     Wd Sdng         0.0        756.0   140000.0
4       2000          2000     VinylSd         0.0       1145.0   250000.0
```

```
dataset.shape
```

```
(2919, 13)
```

In general any ML problem – Maintain Four Pieces

Data	Model	Loss	Optimization

$$\{(x^i, y^i)\}_{i=1}^{N}$$

$$\{(x^i, y^i)\}_{i=1}^{N\text{train}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{val}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{test}}$$

```
obj = (dataset.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:",len(object_cols))

int_ = (dataset.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:",len(num_cols))

fl = (dataset.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:",len(fl_cols))
```

```
Categorical variables: 4
Integer variables: 6
Float variables: 3
```

In general any ML problem – Maintain Four Pieces

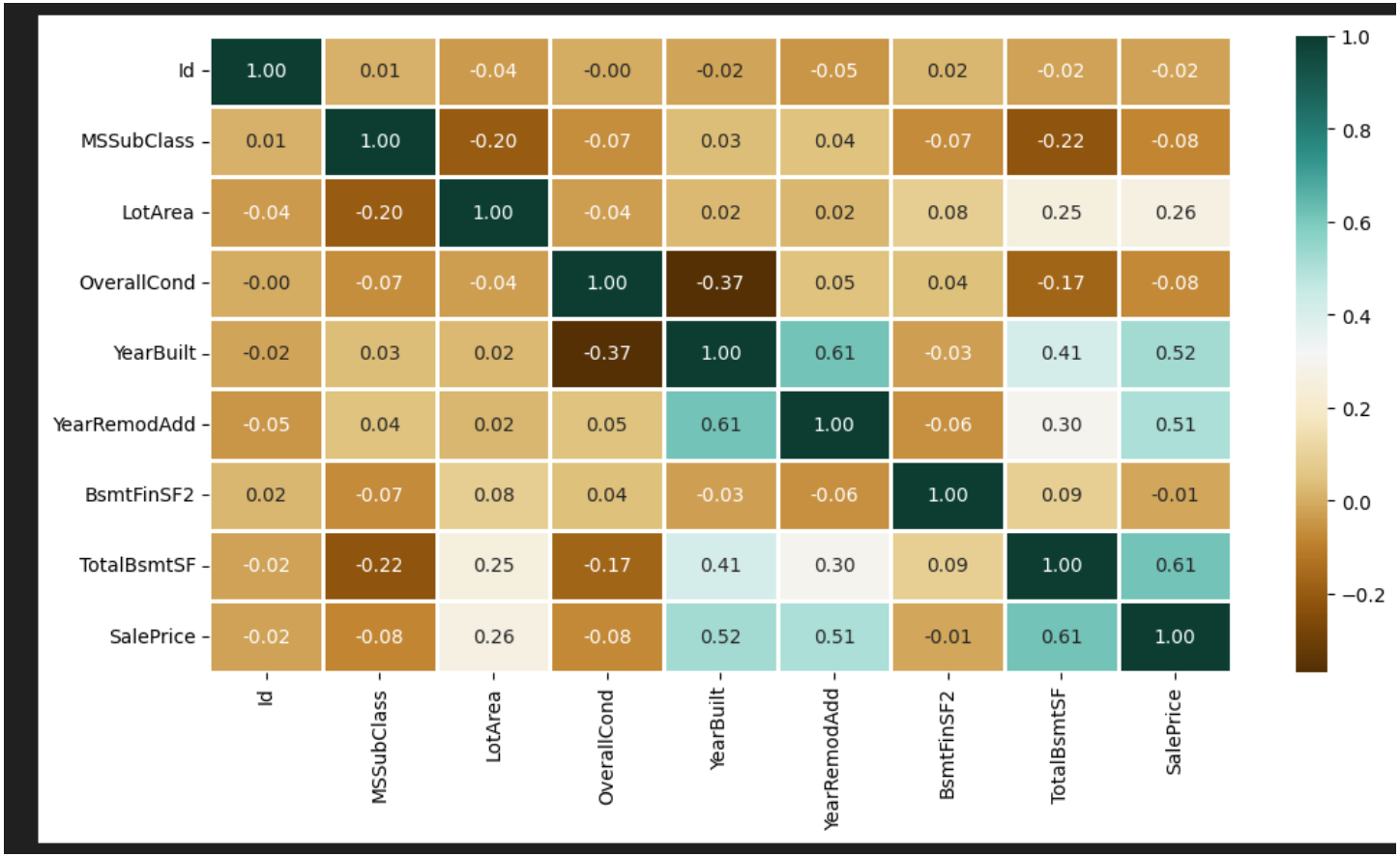
Data	Model	Loss	Optimization

$$\{(x^i, y^i)\}_{i=1}^{N}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{\text{train}}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{val}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{test}}$$



In general any ML problem – Maintain Four Pieces

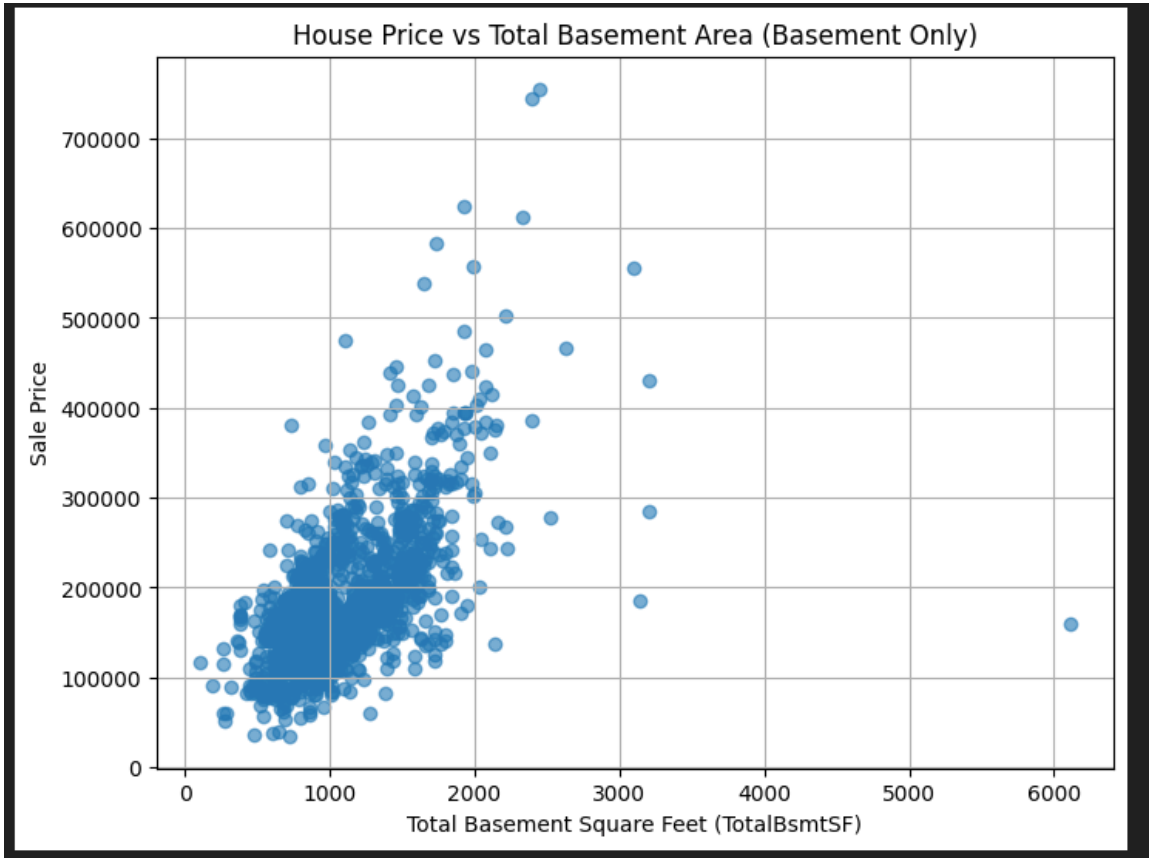
Data | **Model** | **Loss** | **Optimization**

$$\{(x^i, y^i)\}_{i=1}^{N}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{\text{train}}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{val}}$$

$$\{(x^i, y^i)\}_{i=1}^{N_{test}}$$



In general any ML problem – Maintain Four Pieces

Data	**Model**	Loss	Optimization

```
# ------------------------------
# 2) Model
# ------------------------------
model = nn.Sequential(
    nn.Linear(1, 1)   # one-layer neural network (linear regression neuron)
)
```

In general any ML problem – Maintain Four Pieces

Data	Model	Loss	Optimization

```
# ------------------------------
# 3) Loss + Optimizer
# ------------------------------
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
```

In general any ML problem – Maintain Four Pieces

Data | **Model** | **Loss** | **Optimization**

```python
# ----------------------------
# 1) Data
# ----------------------------
X = torch.tensor(X_s, dtype=torch.float32)            # (n,1)
Y = torch.tensor(y,   dtype=torch.float32).view(-1,1) # (n,1)

dataset_t = TensorDataset(X, Y)
loader = DataLoader(dataset_t, batch_size=64, shuffle=True)
```

```python
# ----------------------------
# 4) Train (standard loop) + log optimization process
# ----------------------------
epochs = 50
log_every = 1

loss_history = []

for epoch in range(1, epochs + 1):
    model.train()
    running_loss = 0.0
    n_seen = 0

    for xb, yb in loader:
        optimizer.zero_grad()
        pred = model(xb)
        loss = criterion(pred, yb)
        loss.backward()
        optimizer.step()

        bs = xb.size(0)
        running_loss += loss.item() * bs
        n_seen += bs

    epoch_loss = running_loss / n_seen
    loss_history.append(epoch_loss)

    if log_every and (epoch % log_every == 0):
        print(f"epoch={epoch:3d}  mse={epoch_loss:.4f}")

# Optional: final training MSE on full data
model.eval()
with torch.no_grad():
    final_mse = criterion(model(X), Y).item()
print(f"\nFinal train MSE: {final_mse:.4f}")
```

In general any ML problem – Maintain Four Pieces

Data	Model	Loss	Optimization

```
# ------------------------------
# 3) Loss + Optimizer
# ------------------------------
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
```

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

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$$

In general any ML problem – Maintain Four Pieces

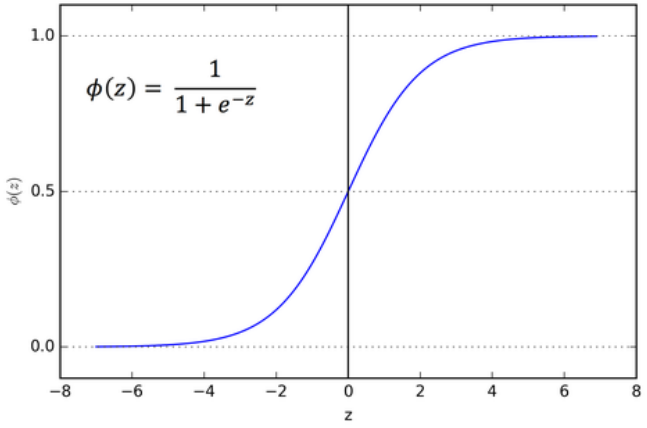
Data	Model	Loss	Optimization

Used for **binary classification** where $y_i \in \{0, 1\}$ and $\hat{y}_i = \sigma(z_i) \in (0, 1)$.

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

Vector form:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{n}\left(\mathbf{y}^T \log \hat{\mathbf{y}} + (1 - \mathbf{y})^T \log(1 - \hat{\mathbf{y}})\right)$$



In general any ML problem – Maintain Four Pieces

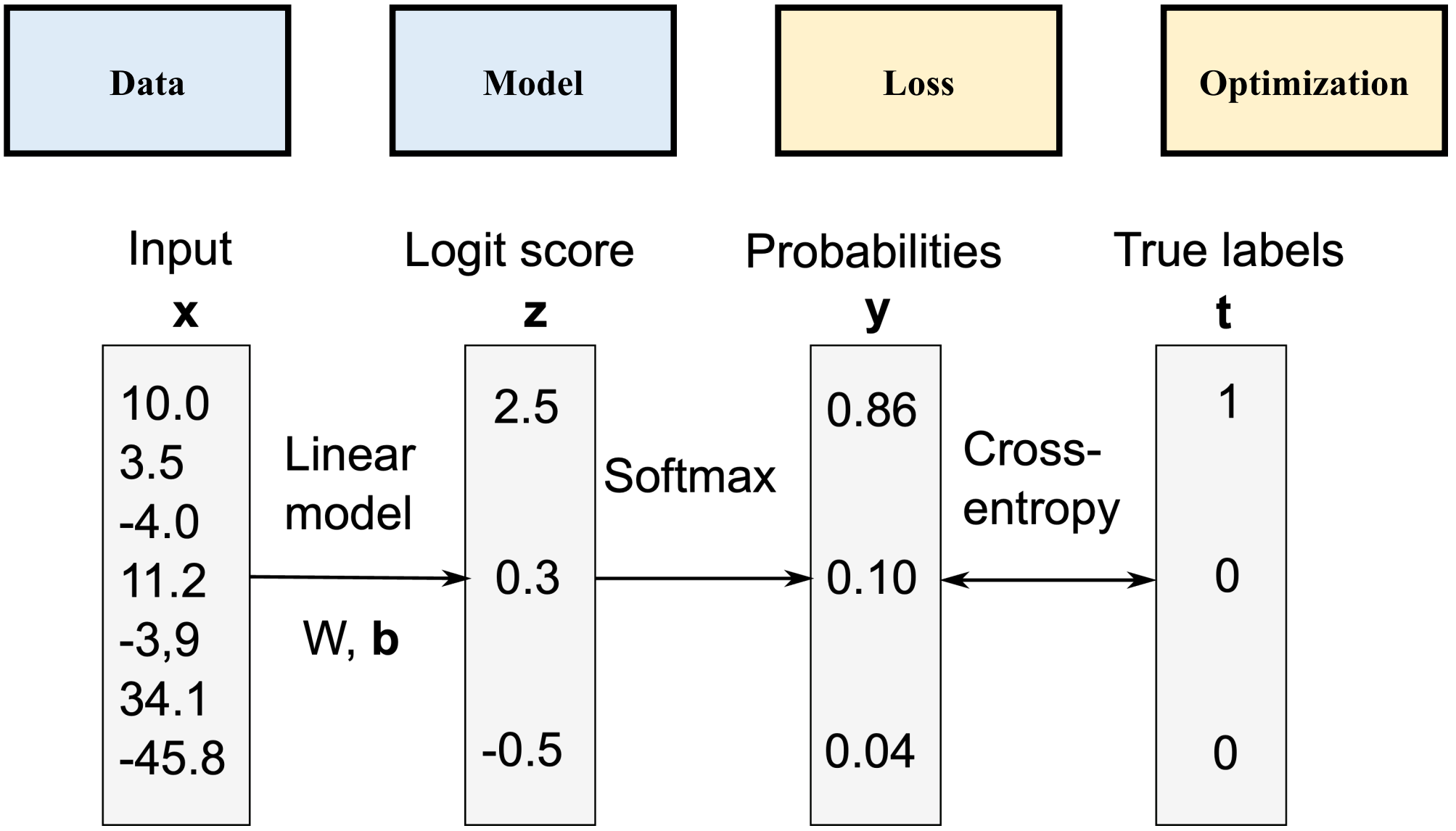
Data	Model	Loss	Optimization

Multi-Class Classification

$$\mathcal{L}_{\text{CE}} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

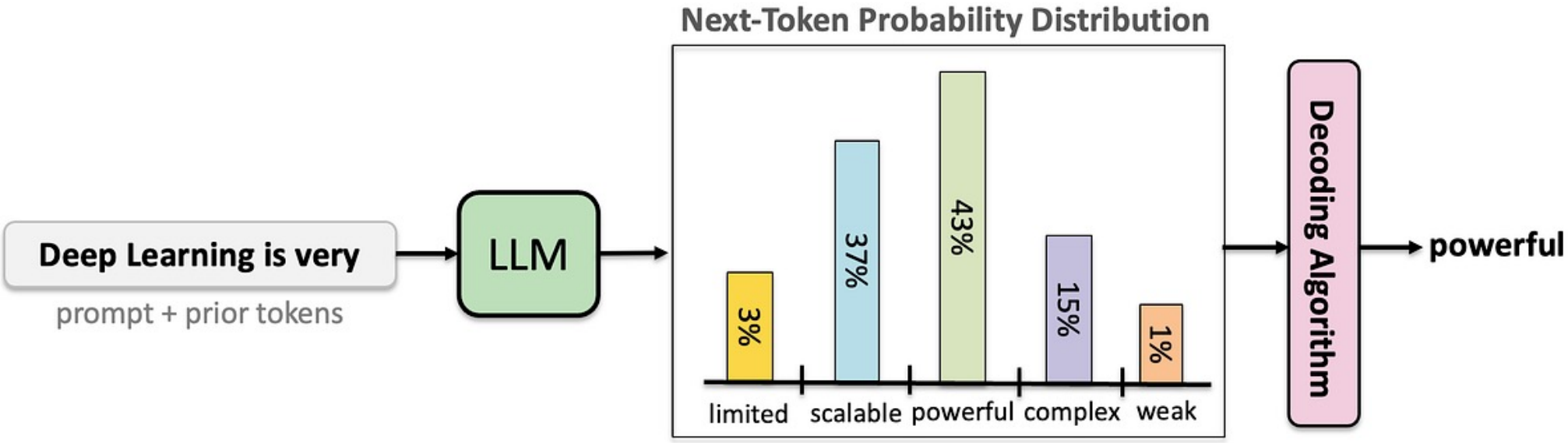
$$\hat{y}_{ic} = \frac{e^{z_{ic}}}{\sum_{k=1}^{C} e^{z_{ik}}}$$

In general any ML problem – Maintain Four Pieces



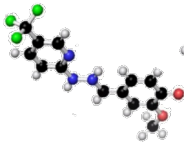
In general any ML problem – Maintain Four Pieces

Data	Model	Loss	Optimization

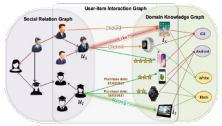


In general any ML problem – Maintain Four Pieces

Data	Model	Loss	Optimization

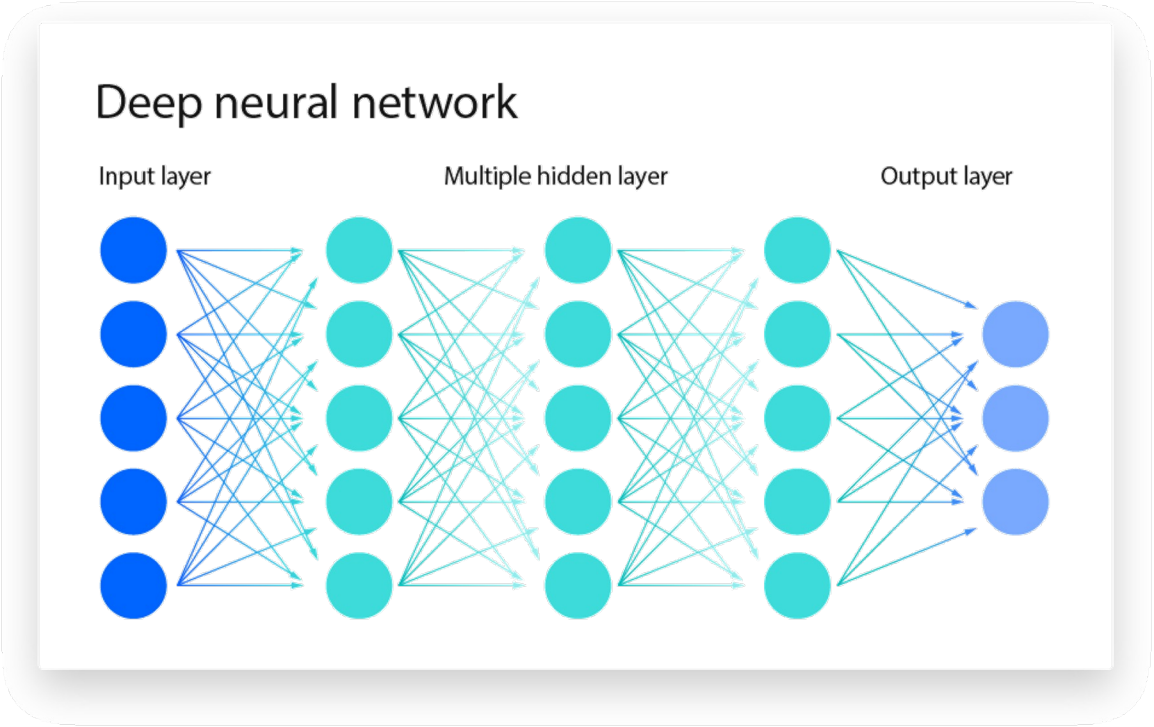


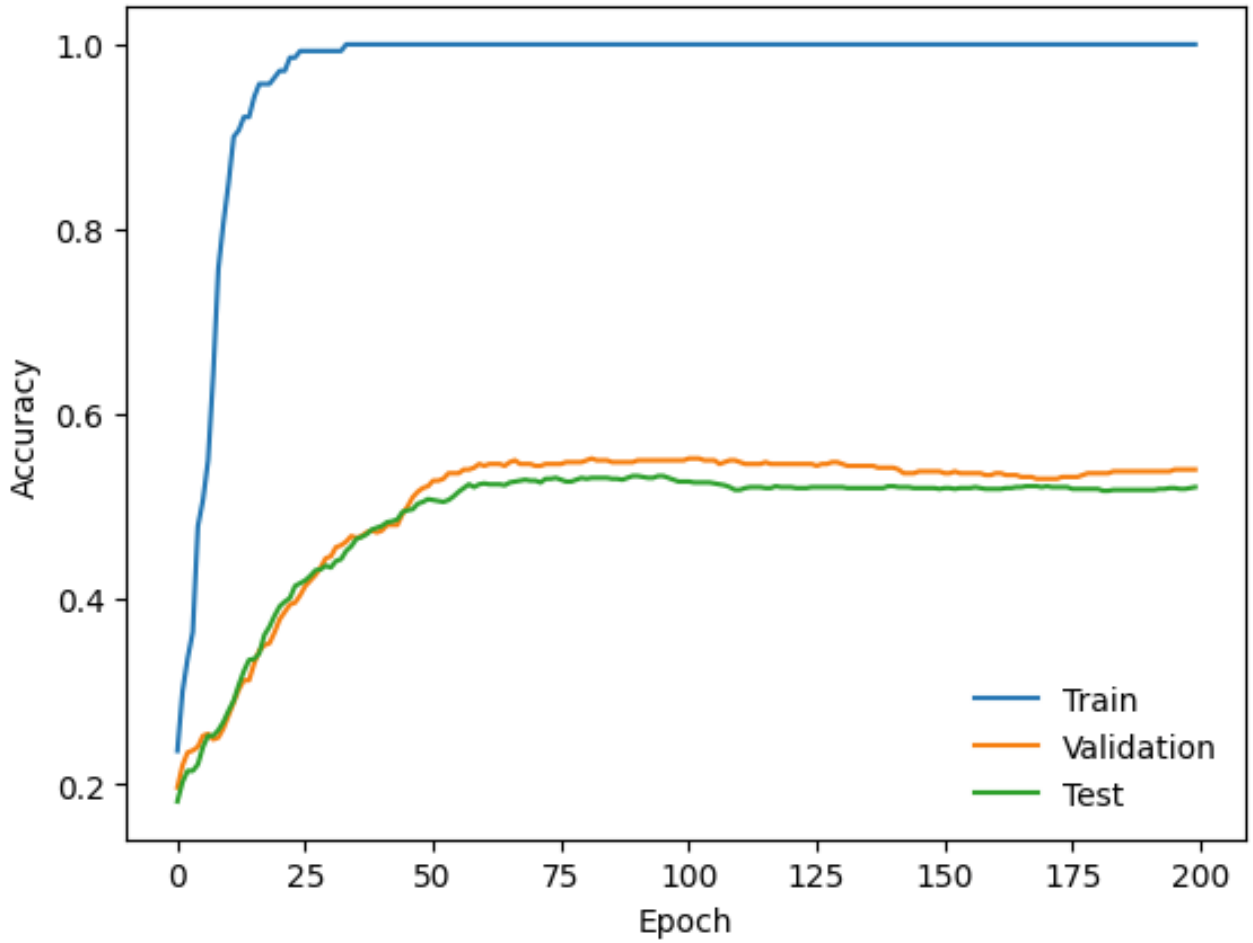
Small Molecule



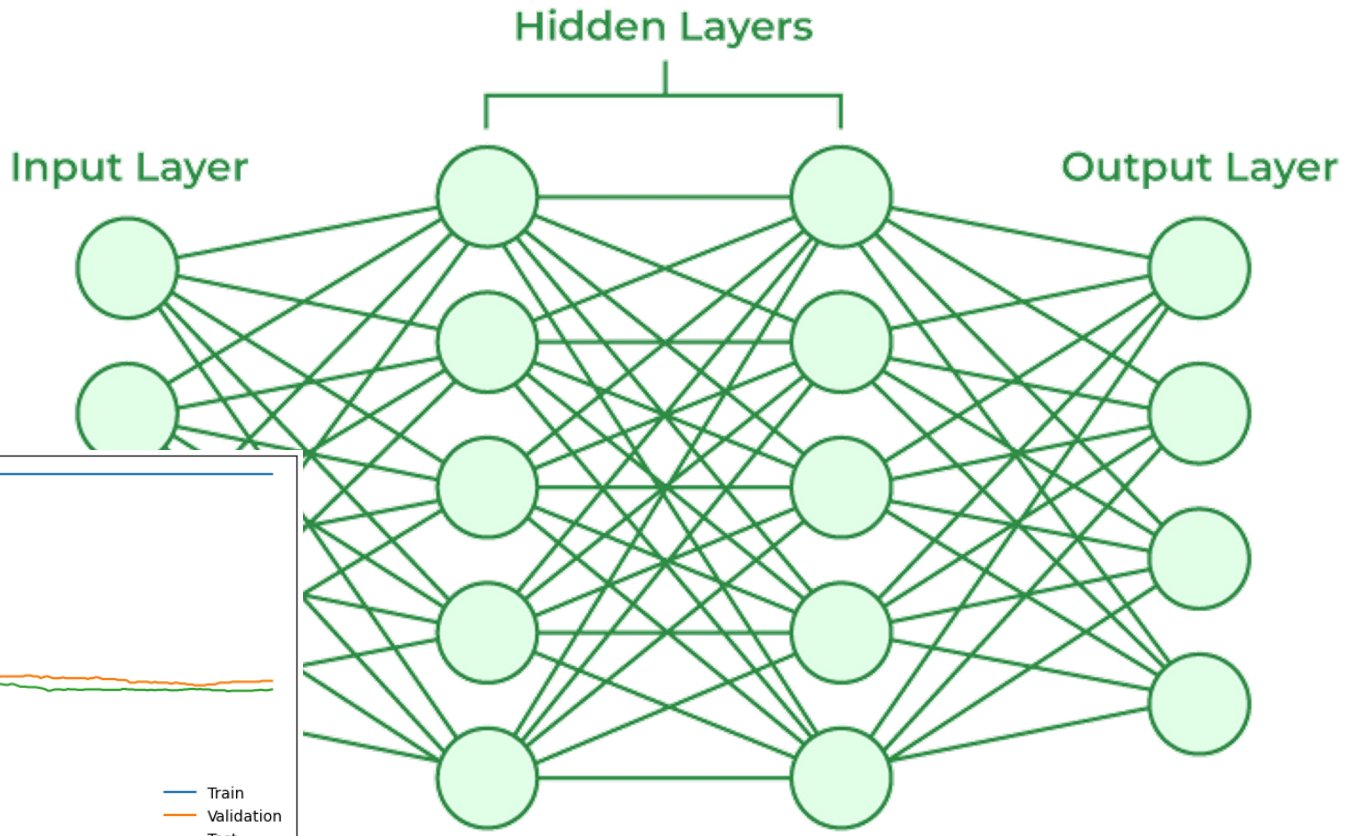
Social Network

Paper

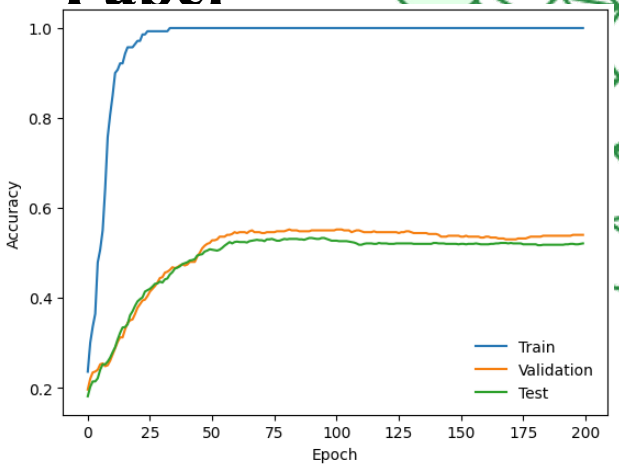




Test Performance:
0.531



Paper



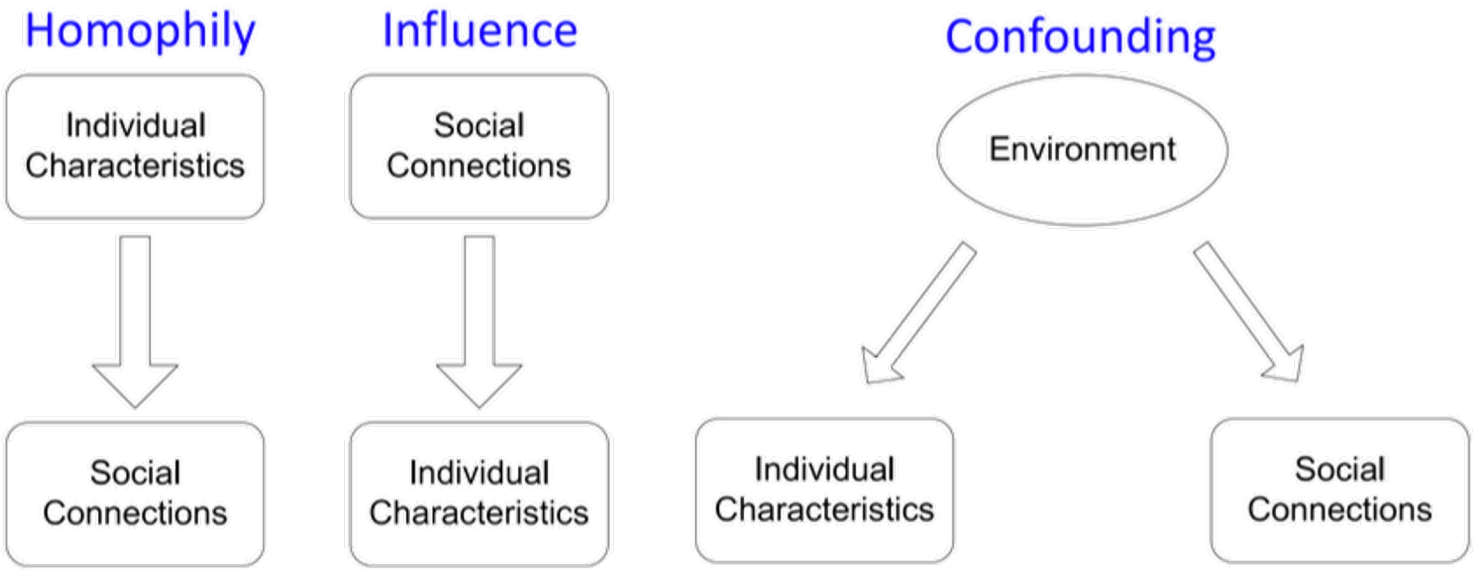
Test Performance: 0.531

Computer System

We do not use any network information!

Correlation in networks

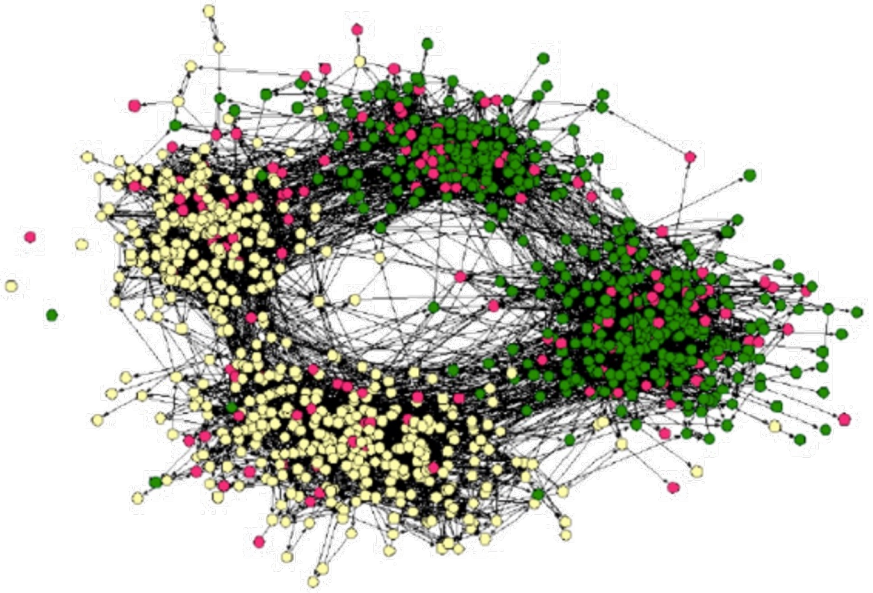
- User behaviors are correlated in social networks

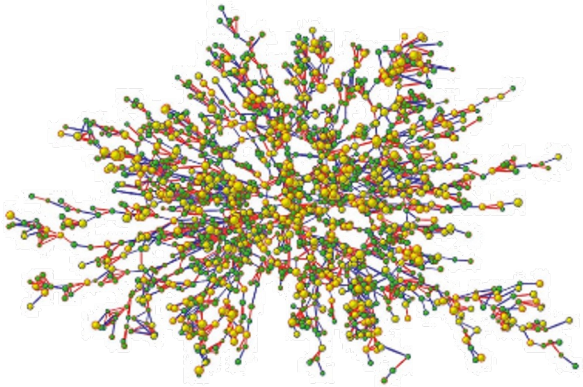


Node Classification - Motivation

Example

- Easley and Kleinberg, 2010.
- Friendships among people in a real social network where node color represents the race of the people/nodes.
- Homophily…

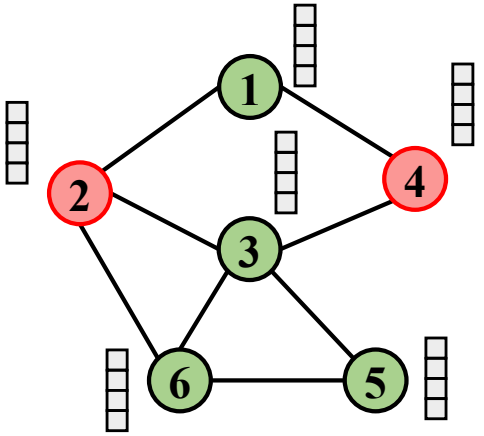




- Christakis and Fowler, 2007.
- Friendships among people in a real social network where node color represents the level of obesity of the people/nodes.
- Influence… and homophily?

Node Classification - How would traditional classification work?

Data | **Model** | **Loss** | **Optimization**




```
from torch_geometric.datasets import Planetoid
```

Dataset

```
dataset = Planetoid(root='/tmp/Cora', name='Cora')
```

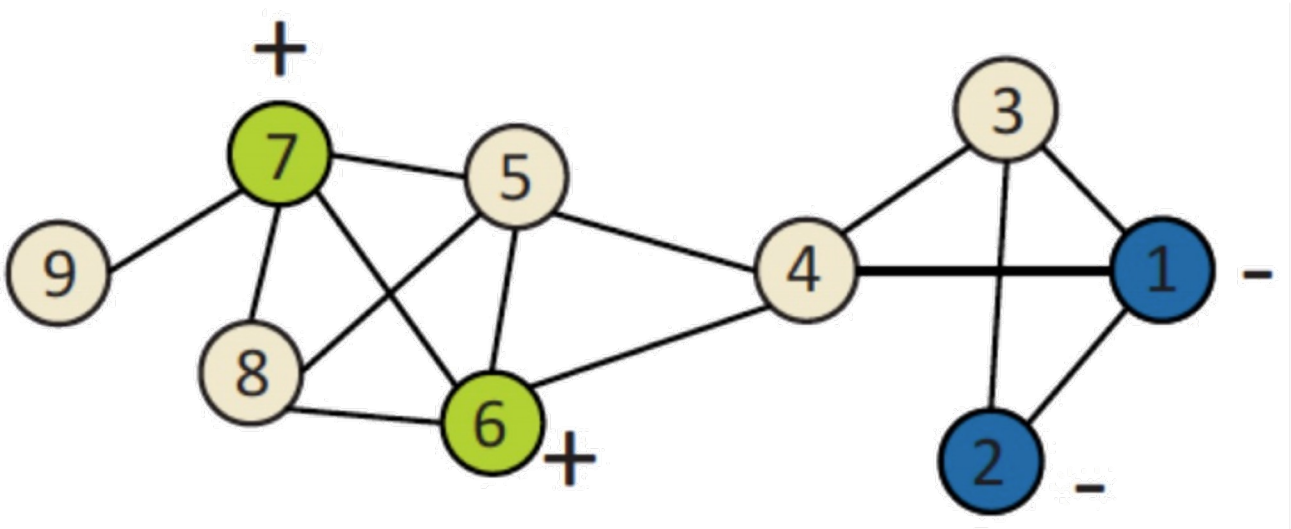
```
dataset[0]
```

```
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
```

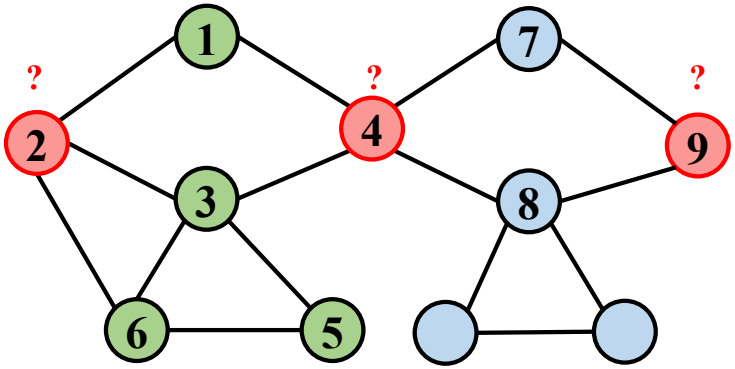
Label Propagation

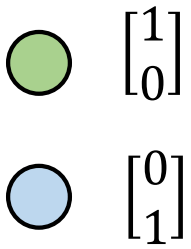
Leveraging this correlation

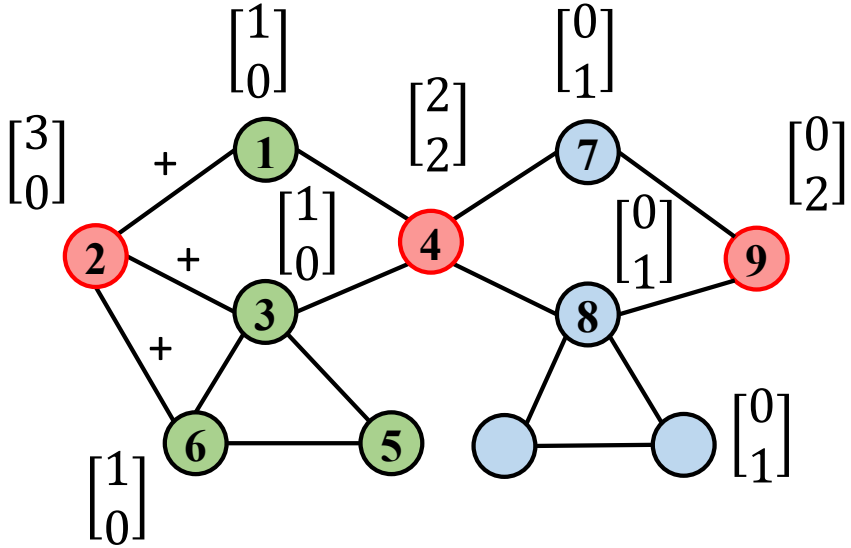
- How to leverage this correlation in networks to predict node classes?

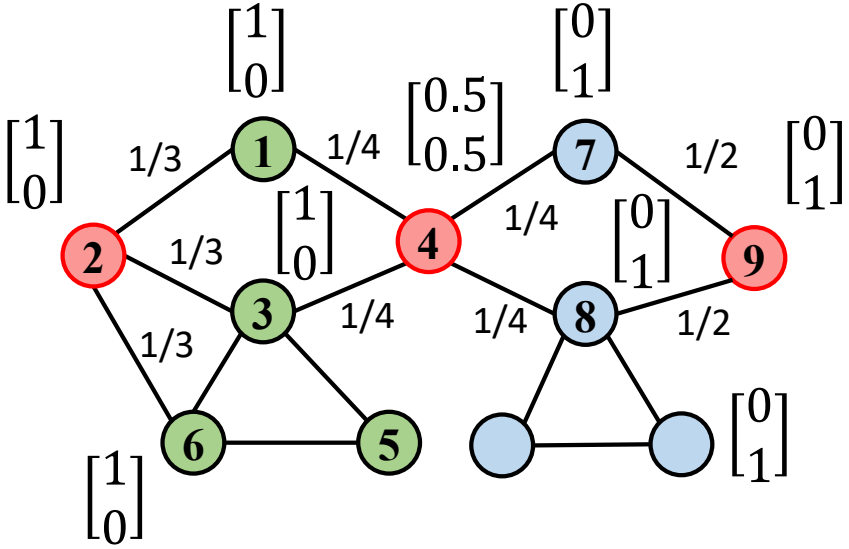


Node Classification – Label Propagation









Node Classification – Label Propagation

```python
from torch_geometric.utils import add_remaining_self_loops, degree
from torch_scatter import scatter

def propagate(x, edge_index, edge_weight=None):
    """ feature propagation procedure: sparsematrix
    """
    edge_index, _ = add_remaining_self_loops(edge_index, num_nodes=x.size(0))


    # calculate the degree normalize term
    row, col = edge_index
    deg = degree(col, x.size(0), dtype=x.dtype)
    deg_inv_sqrt = deg.pow(-0.5)
    # for the first order appro of laplacian matrix in GCN, we use deg_inv_sqrt[row]*deg_inv_sqrt[col]
    if(edge_weight == None):
        edge_weight = deg_inv_sqrt[col] * deg_inv_sqrt[col]


    # normalize the features on the starting point of the edge
    out = edge_weight.view(-1, 1) * x[row]


    return scatter(out, edge_index[-1], dim=0, dim_size=x.size(0), reduce='add')
```

```python
one_hot_label = F.one_hot(data.y, num_classes=torch.unique(data.y).shape[0])
one_hot_label[data.val_mask] = 0
one_hot_label[data.test_mask] = 0

pred = propagate(one_hot_label, data.edge_index)
pred = pred.argmax(dim=1)
train_acc, val_acc, test_acc = cal_acc(pred, data.y, data.train_mask), cal_acc(pred, data.y, data.val_mask), cal_acc(pred, data.y, data.test_mask)
```

```python
print(train_acc, val_acc, test_acc)
```

```
0.9 0.674 0.681
```

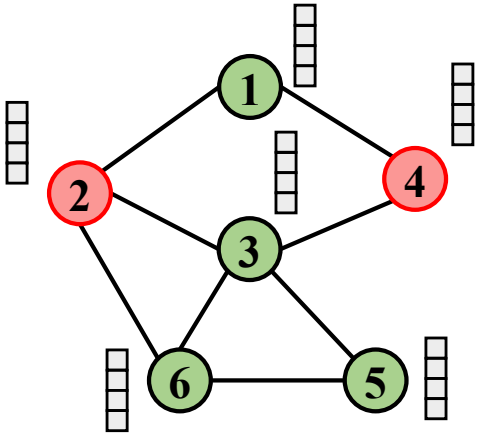
Node Classification

Data | **Model** | **Loss** | **Optimization**

```
from torch_geometric.datasets import Planetoid
```

Dataset

```
dataset = Planetoid(root='/tmp/Cora', name='Cora')
```




```
dataset[0]
```

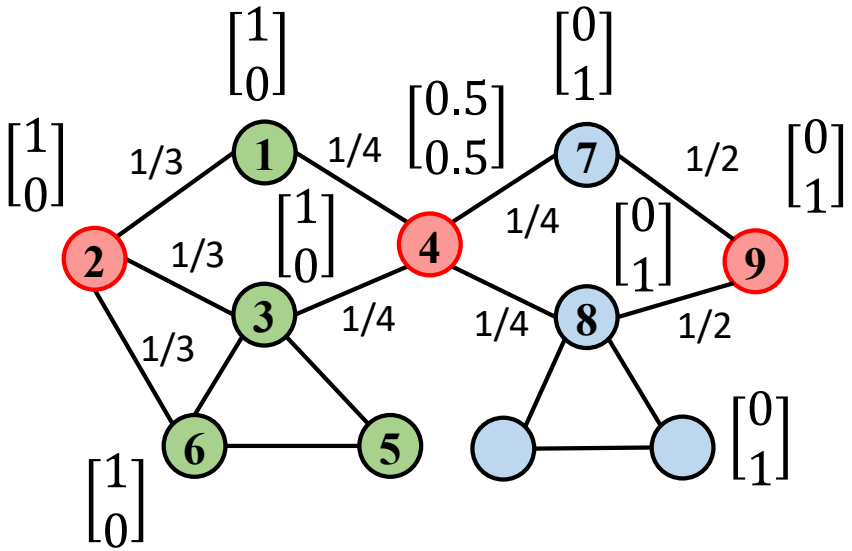
```
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
```

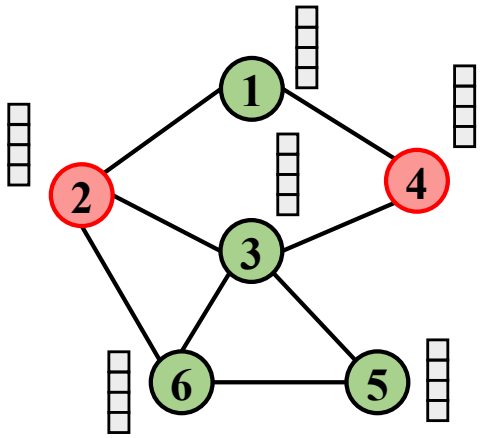
Feature + **Label** = **MLP**

Graph + **Label** = **Label Propagation**

Feature + **Label** + **Graph** = **GNN**

Node Classification - GNN



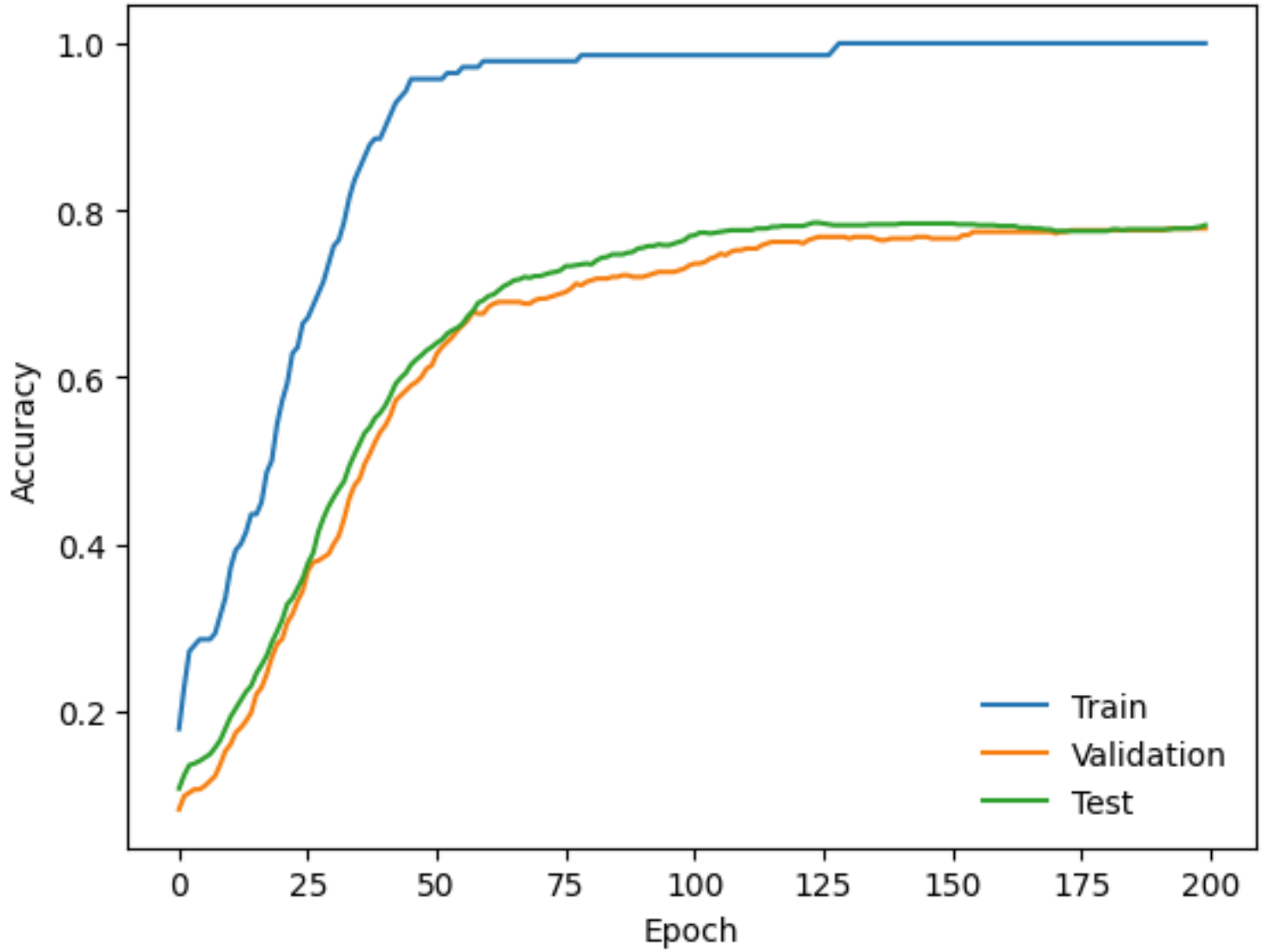


Instead of propagating Label, but we propagate feature!

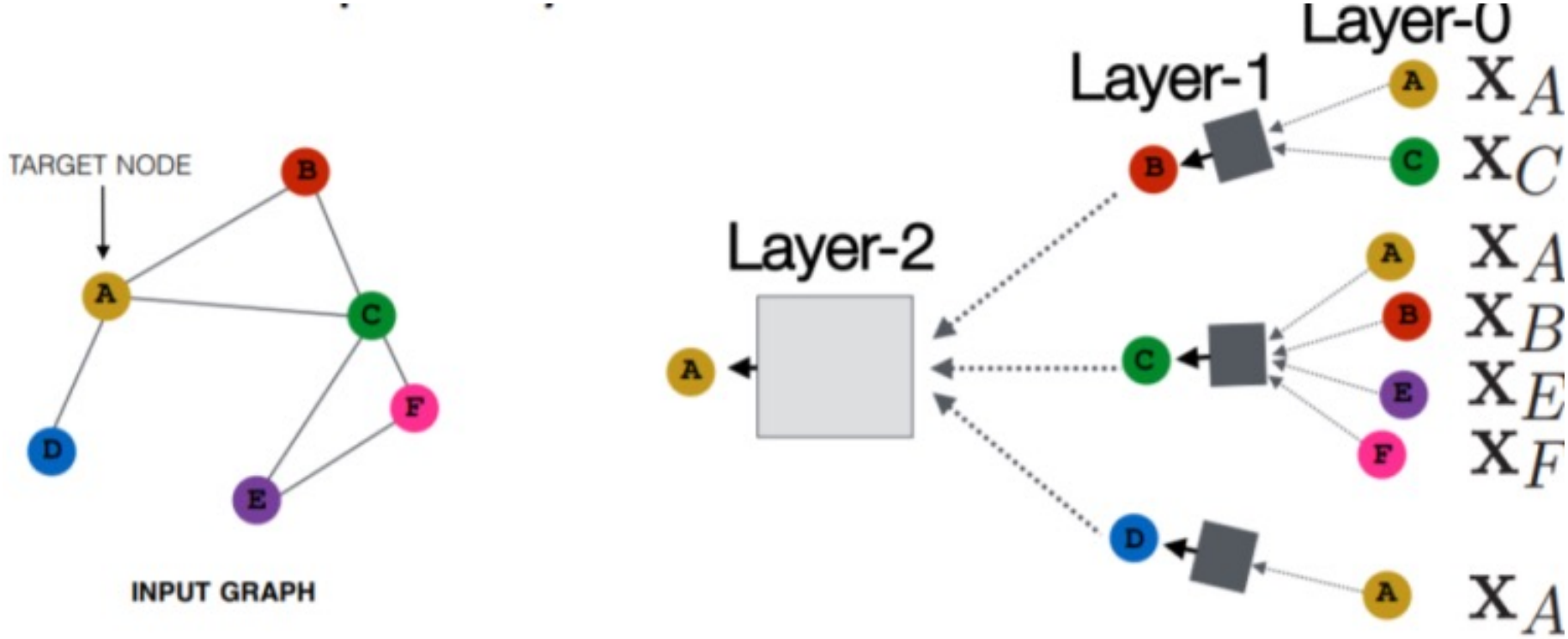
Node Classification – GNN

```python
class GCN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(GCN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x, edge_index, edge_weight = None):
        x = self.fc1(x)
        x = propagate(x, edge_index)
        x = self.relu(x)
        # x = F.dropout(x, p=0.5, training=self.training)
        x = self.fc2(x)
        x = propagate(x, edge_index)

        return F.log_softmax(x, dim=1)

# Example usage:
input_size = dataset[0].x.shape[1]
hidden_size = 16
output_size = torch.unique(dataset[0].y).shape[0]

GNN = GCN(input_size, hidden_size, output_size)
print(model)
```

Node Classification – GNN



Node Classification – GNN



$\mathbb{R}^{6\times 6}$ $\mathbb{R}^{1433\times 16}$

$$\mathbf{H^1 = AH^0W}$$

$\mathbb{R}^{6\times 16}$

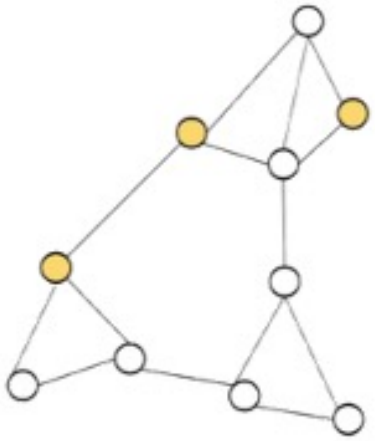
$\mathbb{R}^{6\times 1433}$

Node Classification

Data	Model	Loss	Optimization

Labeled node v_i with label y_i

Unlabeled node



Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}) = (\mathbf{A}, \mathbf{X})$

Adjacency Matrix $\mathbf{A}$

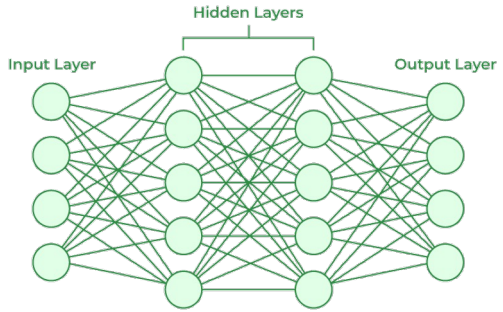
Node Feature Matrix $\mathbf{X}$

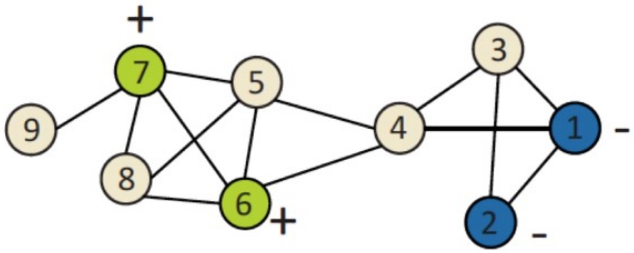
Labeled Data $\mathcal{D}_L = (\mathcal{V}_L, \mathcal{Y}_L)$

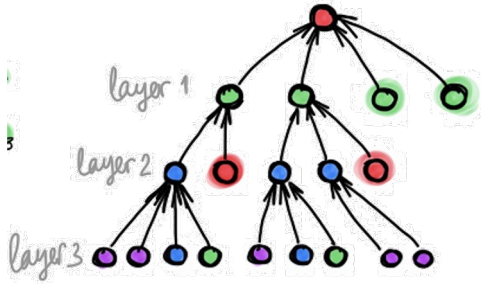
Unlabeled Data $\mathcal{D}_U = (\mathcal{V}_U)$

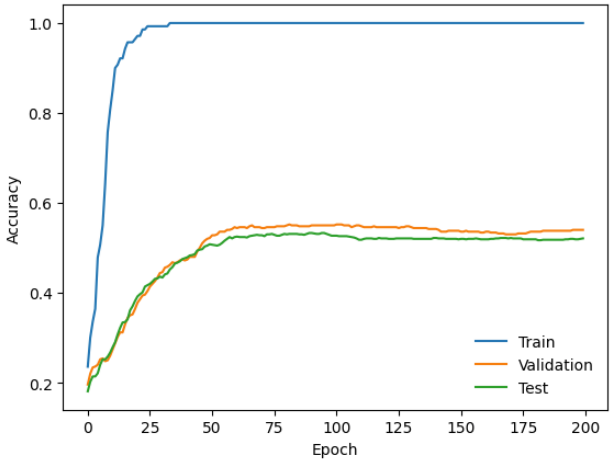
Node Classification

Data | **Model** | **Loss** | **Optimization**



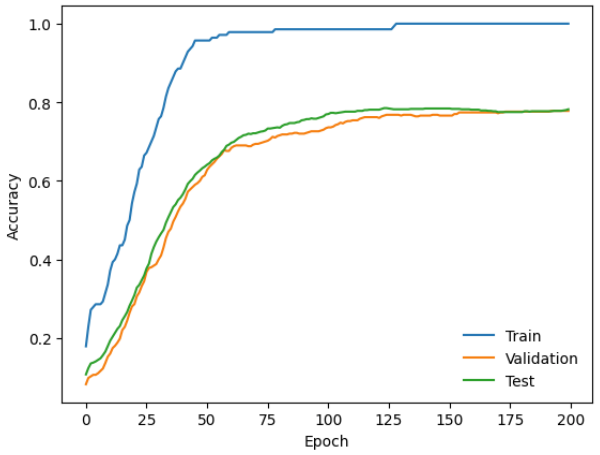







```
print(train_acc, val_acc, test_acc)
```

```
0.9 0.674 0.681
```

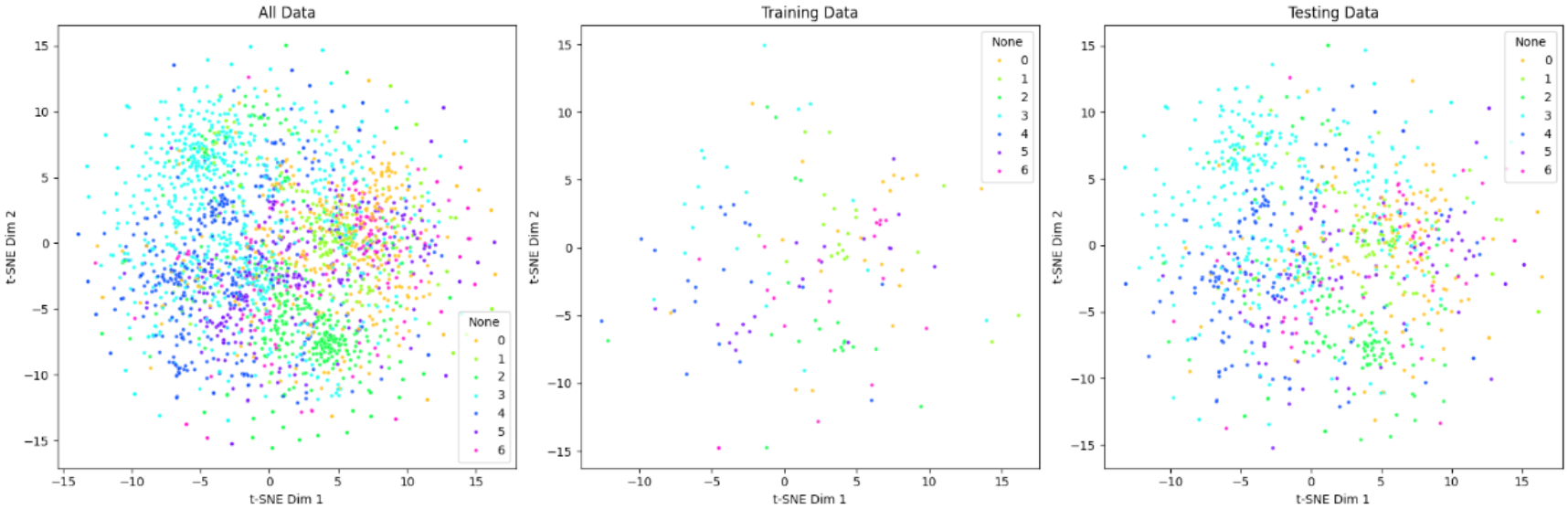


Test Performance: 0.531

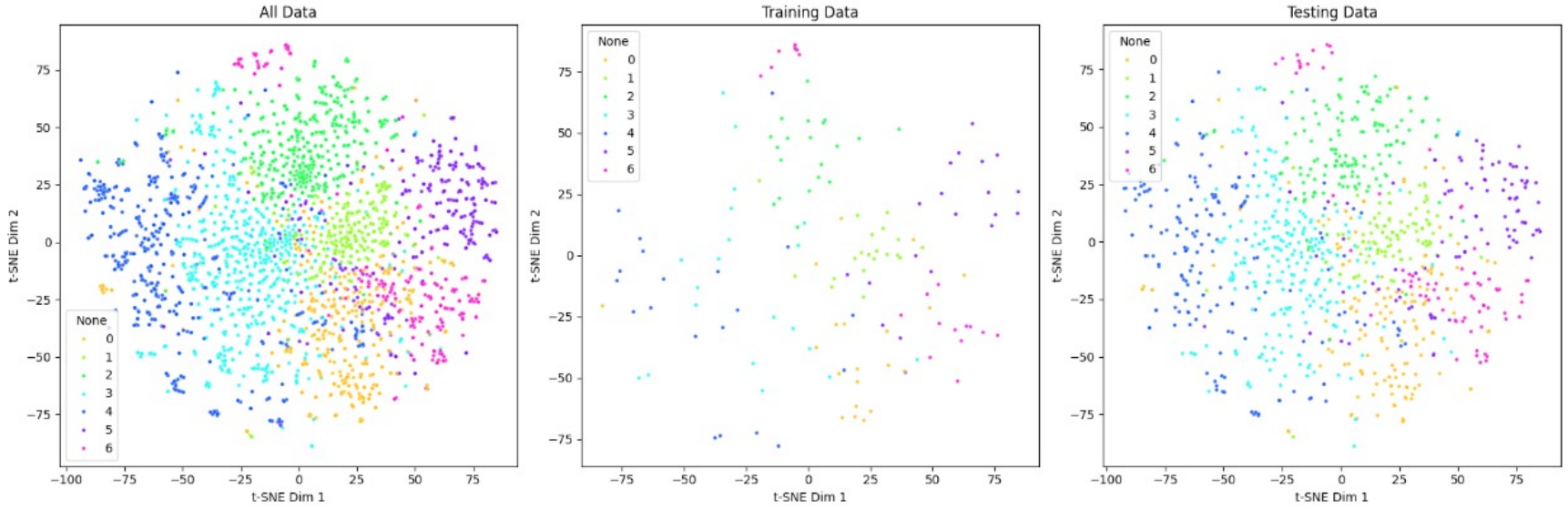
Test Performance: 0.788

Node Classification – Input Space

MLP

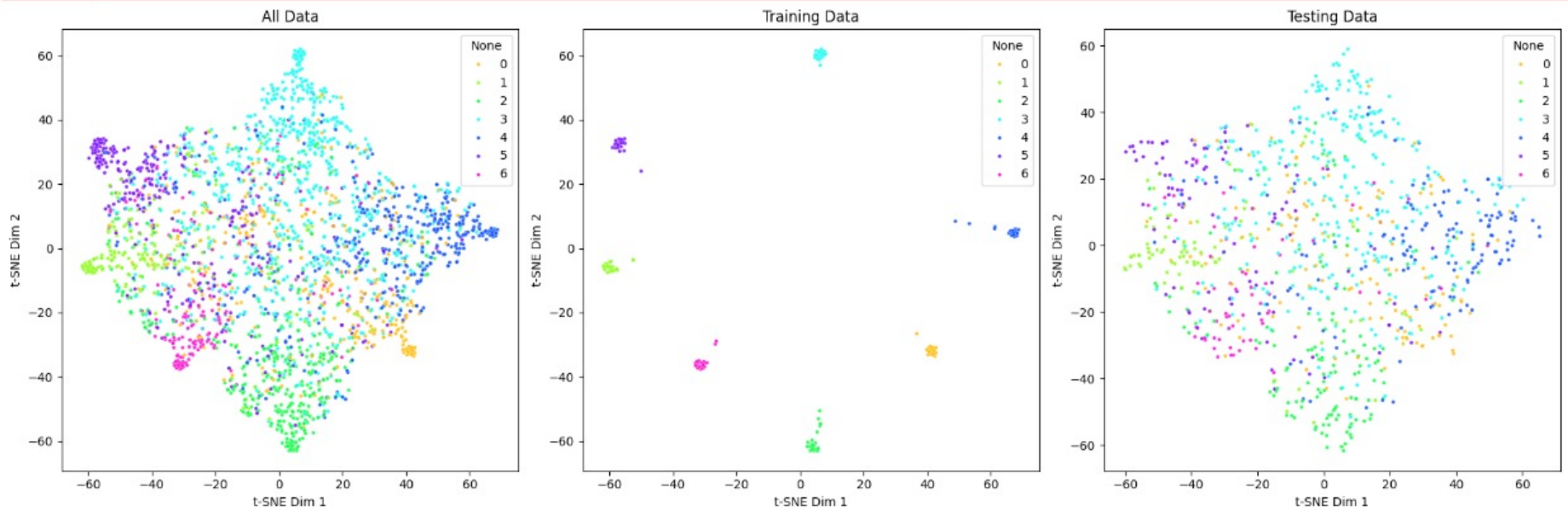


GNN

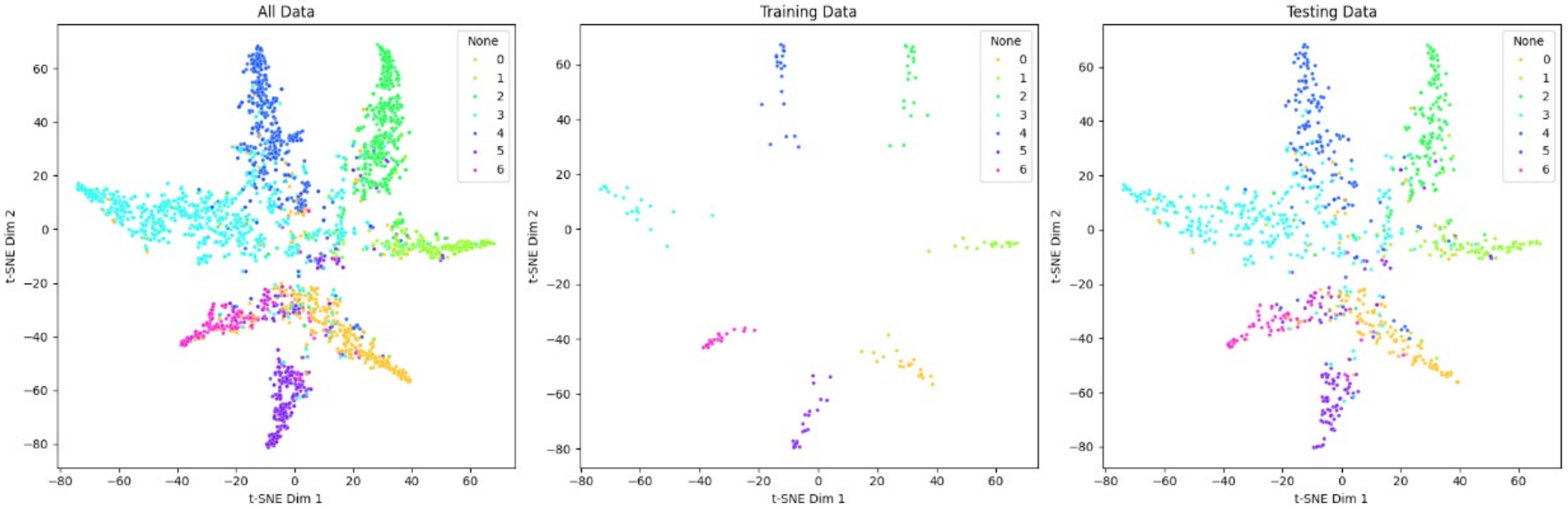


Node Classification – Embedding Space

MLP

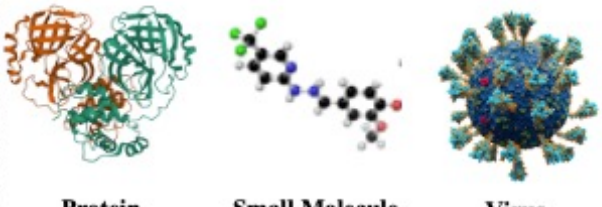


GNN



Link Prediction - Problem

Scientific Graph



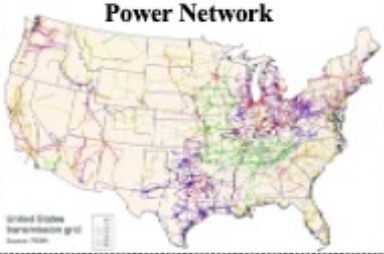
Protein | Small Molecule | Virus

Brain Neural | Phylogenetic Tree | 3D Grid

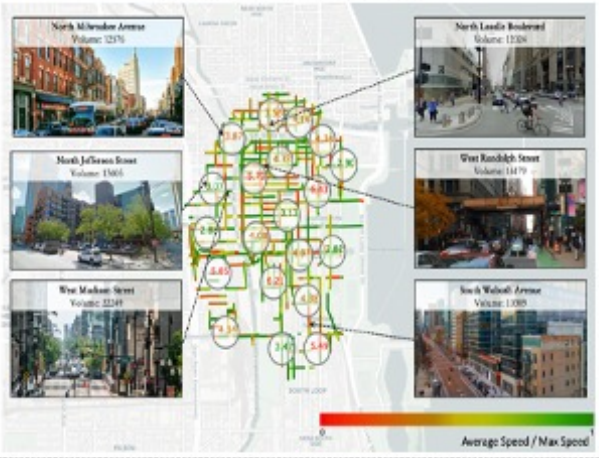
Infrastructure Graph

Gas Network

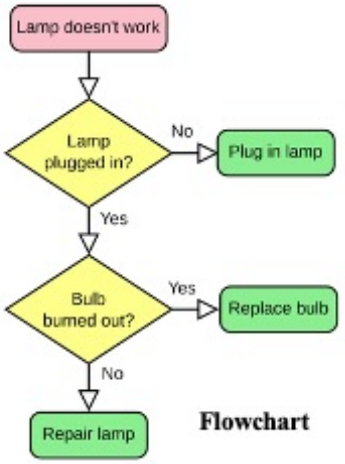
Power Network



Transportation Network



Decision Graph



Flowchart

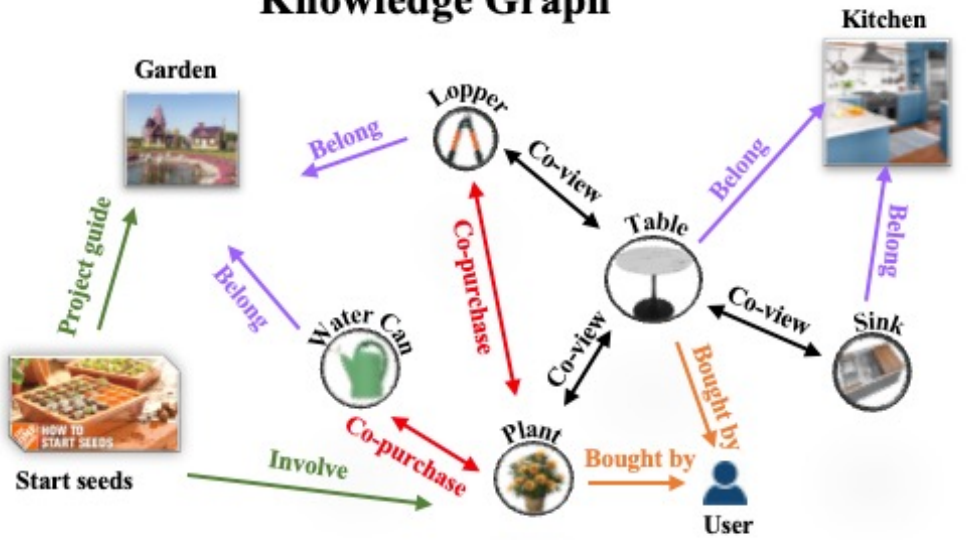
Social Interaction Graph

Citation Network | Transaction Network

User-Entity Interaction Graph

Virtual Village with AI Agents

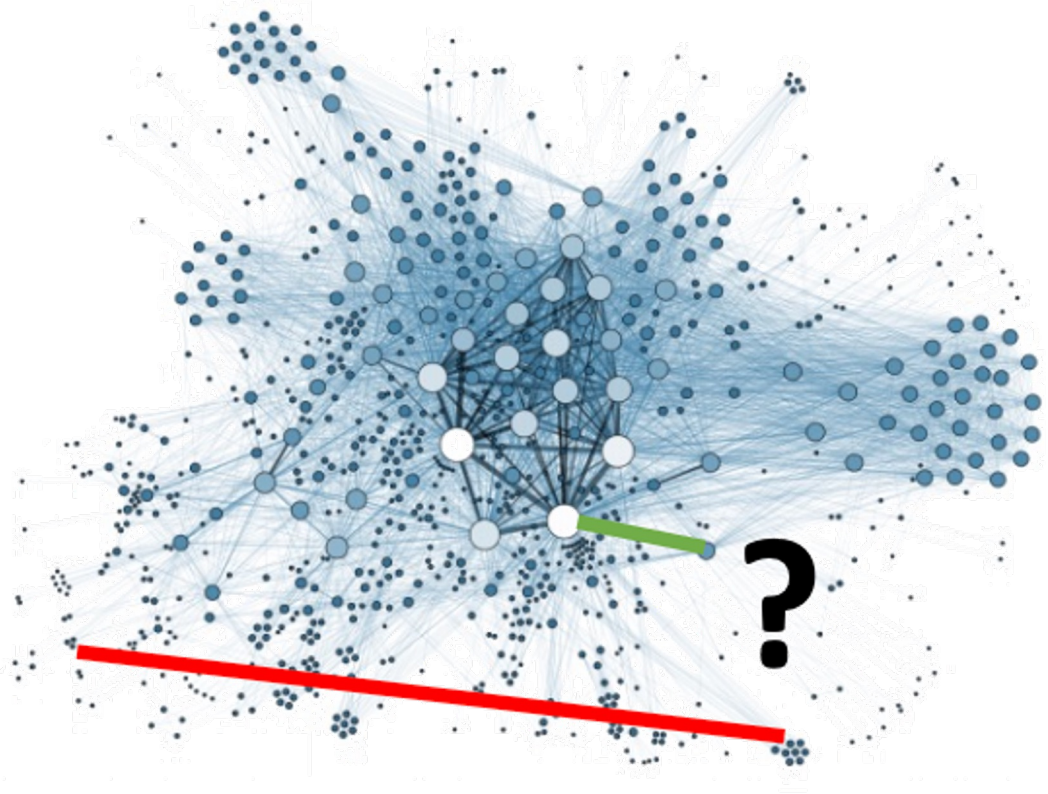
Knowledge Graph



The Home Depot Product Knowledge Graph

Link Prediction

- Which nodes in the graph that are not currently connected should be, or perhaps missing?

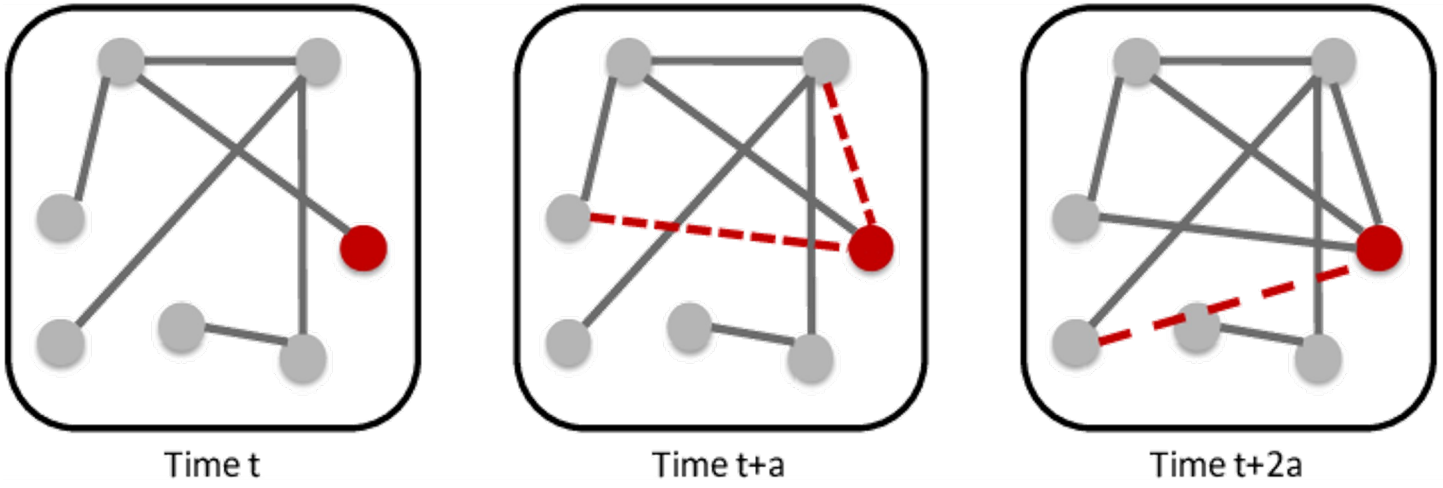


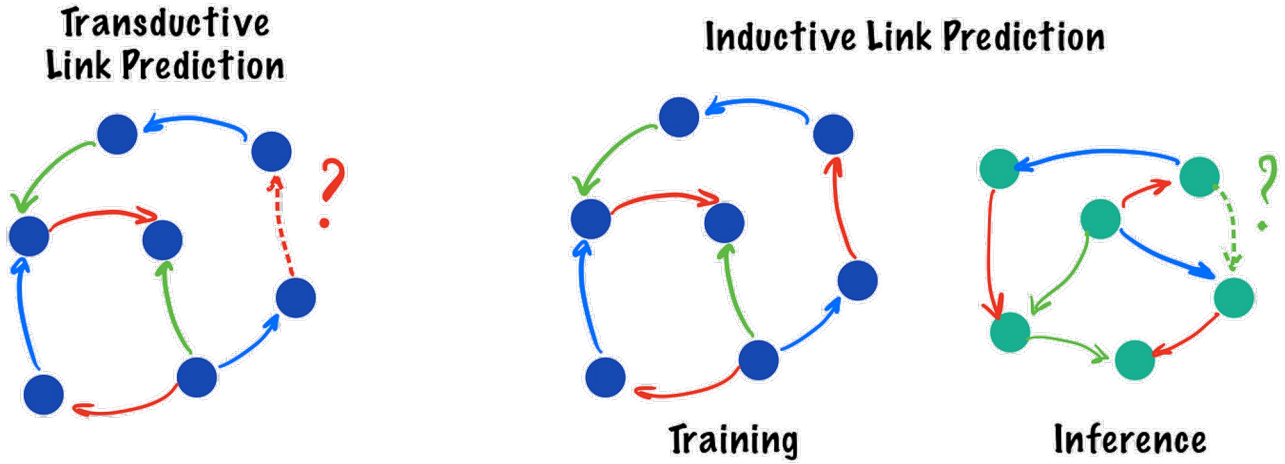
Link Prediction - Problem

- Link prediction: We have a network changing over time and given a snapshot of the network at some time can we predict edges added by the next time step

- What do we exactly predict?
  - Link existence (i.e., binary classification)
  - Link weight (i.e., how strong is that connection)
  - Link type (e.g., is this a positive or negative relationship)

Link Prediction - Problem





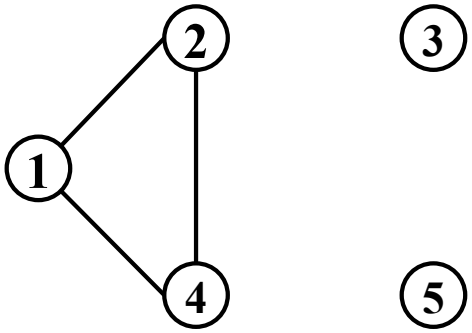
Link Prediction - Data

```
data
```

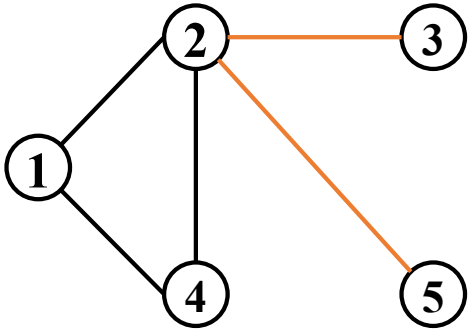
```
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708], train_edge_index=[2, 7392])
```

```
split_edge
```

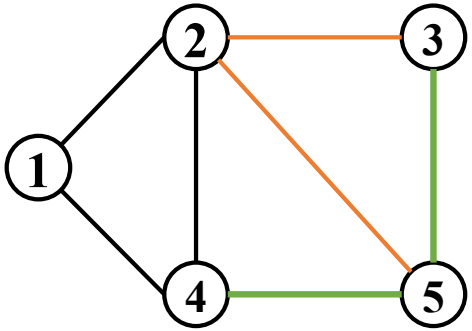

```
{'train': {'edge': tensor([[   0, 1862],
          [   0, 2582],
          [   1,    2],
          ...,
          [2687, 2688],
          [2694, 2695],
          [2706, 2707]]),
  'edge_neg': tensor([[   1, 1457],
          [   1, 2045],
          [   2,  175],
          ...,
          [2593, 2687],
          [2622, 2649],
          [2634, 2695]])},
 'valid': {'edge': tensor([[   1,  654],
          [   4, 1256],
          [   4, 2176],
          ...,
          [2565, 2567],
          [2594, 2680],
          [2665, 2666]]),
  'edge_neg': tensor([[   1, 1457],
          [   1, 2045],
          [   2,  175],
          ...,
          [2593, 2687],
          [2622, 2649],
          [2634, 2695]])},
 'test': {'edge': tensor([[   0,  633],
          [   4, 1016],
          [   6,  373],
          ...,
          [2618, 2619],
          [2634, 2635],
          [2654, 2655]]),
  'edge_neg': tensor([[   0, 1211],
          [   0, 1571],
          [   1,  474],
          ...,
          [2572, 2595],
          [2585, 2672],
          [2619, 2674]])}}
```

Train-edge



Val-edge

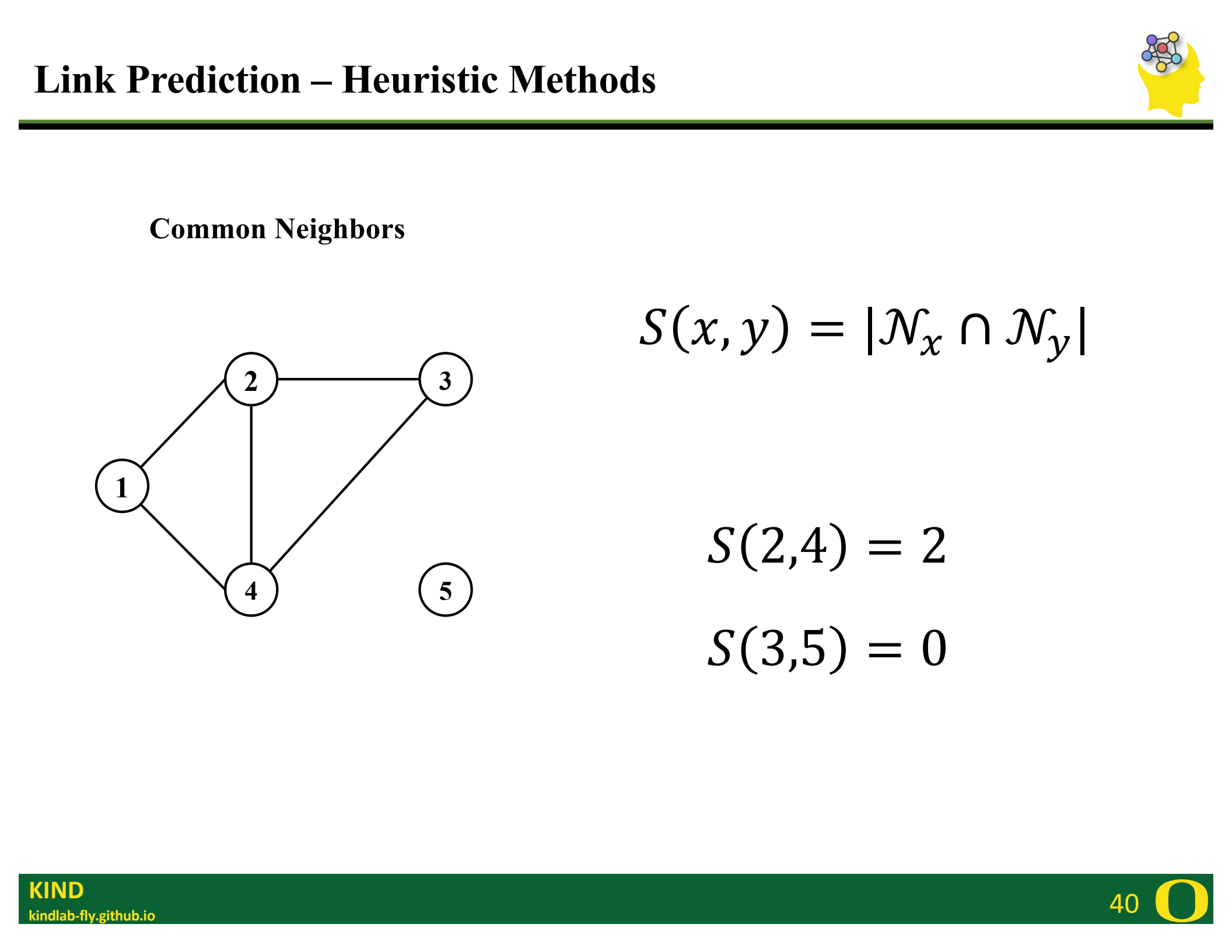


Test-edge

Link Prediction and Recommendation Systems

1. **Heuristic Method**
2. **MLP**
3. **Graph Neural Network**

Link Prediction – Heuristic Methods



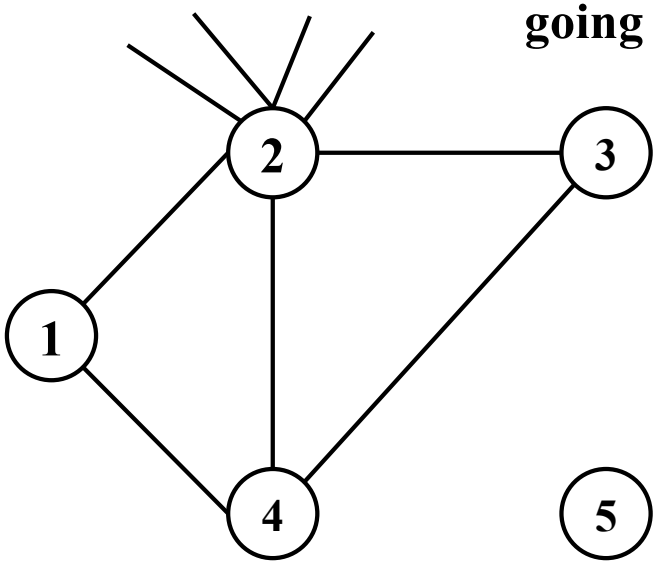
Common Neighbors

$$S(x, y) = |\mathcal{N}_x \cap \mathcal{N}_y|$$

$$S(2,4) = 2$$

$$S(3,5) = 0$$

Link Prediction – Heuristic Methods



Person 2 is a very out-going person

$$S(2,4) = 2$$

The common connection between 2 and 4 might not due to their common hobbies but just because node 2 is an out-going person

$$S(x,y) = \frac{|\mathcal{N}_x \cap \mathcal{N}_y|}{|\mathcal{N}_x \cup \mathcal{N}_y|}$$

Link Prediction – Heuristic Methods

```
nei_dict = defaultdict(list)
for i in range(split_edge['train']['edge'].shape[0]):
    node1, node2 = split_edge['train']['edge'][i][0].item(), split_edge['train']['edge'][i][1].item()
    nei_dict[node1].append(node2)
    nei_dict[node2].append(node1)
```

```
def common_nei_heuristic(nei1, nei2):
    return len(set(nei1).intersection(set(nei2)))

def jaccard_heuristic(nei1, nei2):
    if len(set(nei1).union(set(nei2))) == 0:
        return 0
    else:
        return common_nei_heuristic(nei1, nei2)/len(set(nei1).union(set(nei2)))
```

Link Prediction – Heuristic Methods

```python
nei_dict = defaultdict(list)
for i in range(split_edge['train']['edge'].shape[0]):
    node1, node2 = split_edge['train']['edge'][i][0].item(), split_edge['train']['edge'][i][1].item()
    nei_dict[node1].append(node2)
    nei_dict[node2].append(node1)
```

```python
def common_nei_heuristic(nei1, nei2):
    return len(set(nei1).intersection(set(nei2)))

def jaccard_heuristic(nei1, nei2):
    if len(set(nei1).union(set(nei2))) == 0:
        return 0
    else:
        return common_nei_heuristic(nei1, nei2)/len(set(nei1).union(set(nei2)))
```

```
{'train': [0.33766233766233766],
 'valid': [0.3187855787476281],
 'test': [0.3345971563981043]}
```

```python
def eval(evaluator, split_edge, nei_dict, topks = [20]):
    encoder.eval()
    predictor.eval()

    ress = {'train': [],
            'valid': [],
            'test': []}


    # Evaluation per edge
    for key in split_edge:
        edge, neg_edge = split_edge[key]['edge'], split_edge[key]['edge_neg']

        pos_preds = []

        for node1, node2 in edge:
            nei1, nei2 = nei_dict[node1.item()], nei_dict[node2.item()]

            common_nei = common_nei_heuristic(nei1, nei2)
            pos_preds.append(common_nei)

        pos_preds = torch.tensor(pos_preds)

        print(pos_preds)
        neg_preds = []
        for node1, node2 in neg_edge:
            nei1, nei2 = nei_dict[node1.item()], nei_dict[node2.item()]

            common_nei = common_nei_heuristic(nei1, nei2)
            neg_preds.append(common_nei)

        neg_preds = torch.tensor(neg_preds)

        # Evaluate for each K in top-k list
        for K in topks:
            evaluator.K = K
            hits = evaluator.eval({'y_pred_pos': pos_preds, 'y_pred_neg': neg_preds})[f'hits@{K}']
            ress[key].append(hits)


    return ress
```

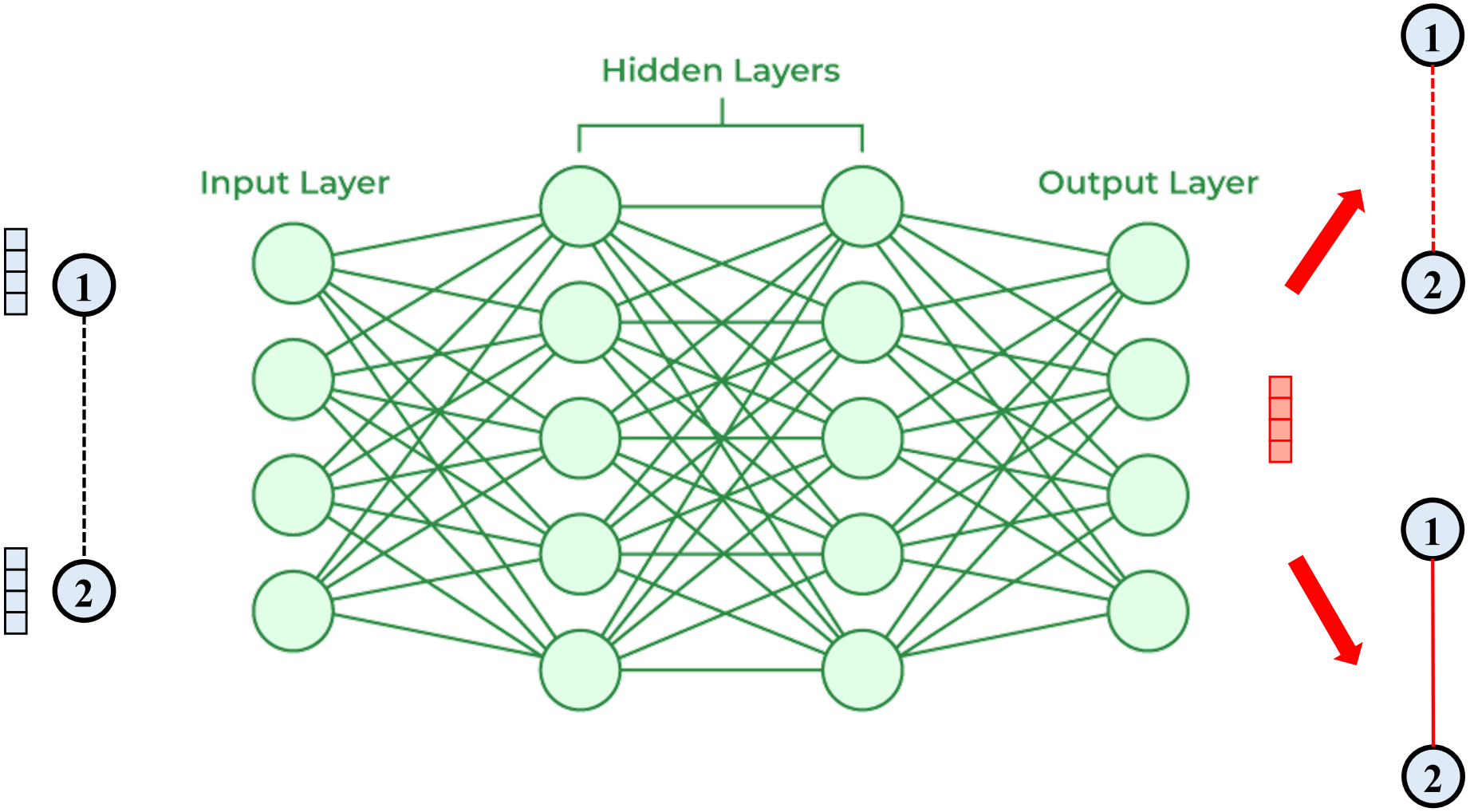
Link Prediction – Heuristic Methods

1. Heuristic Method

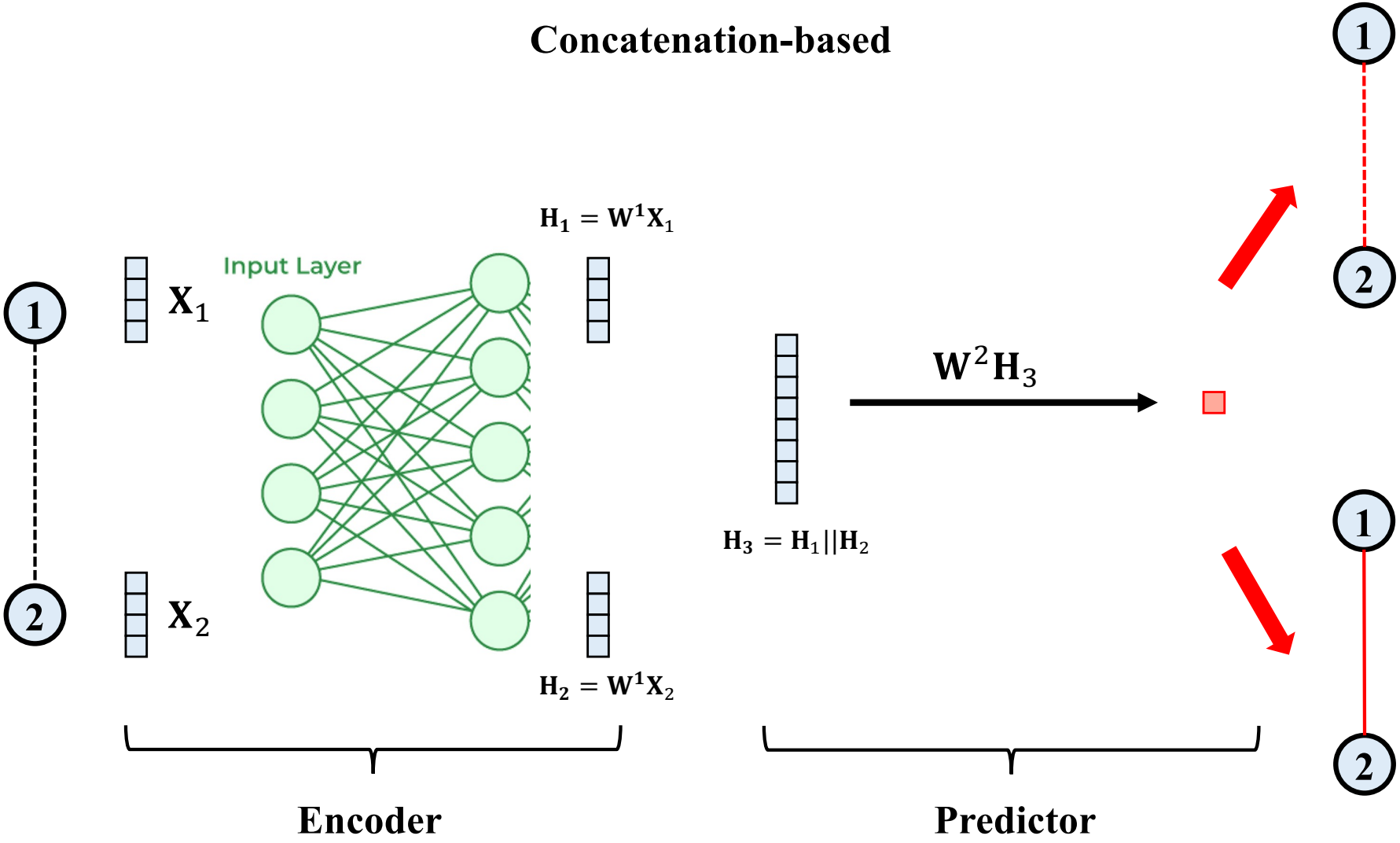
2. MLP

3. Graph Neural Network

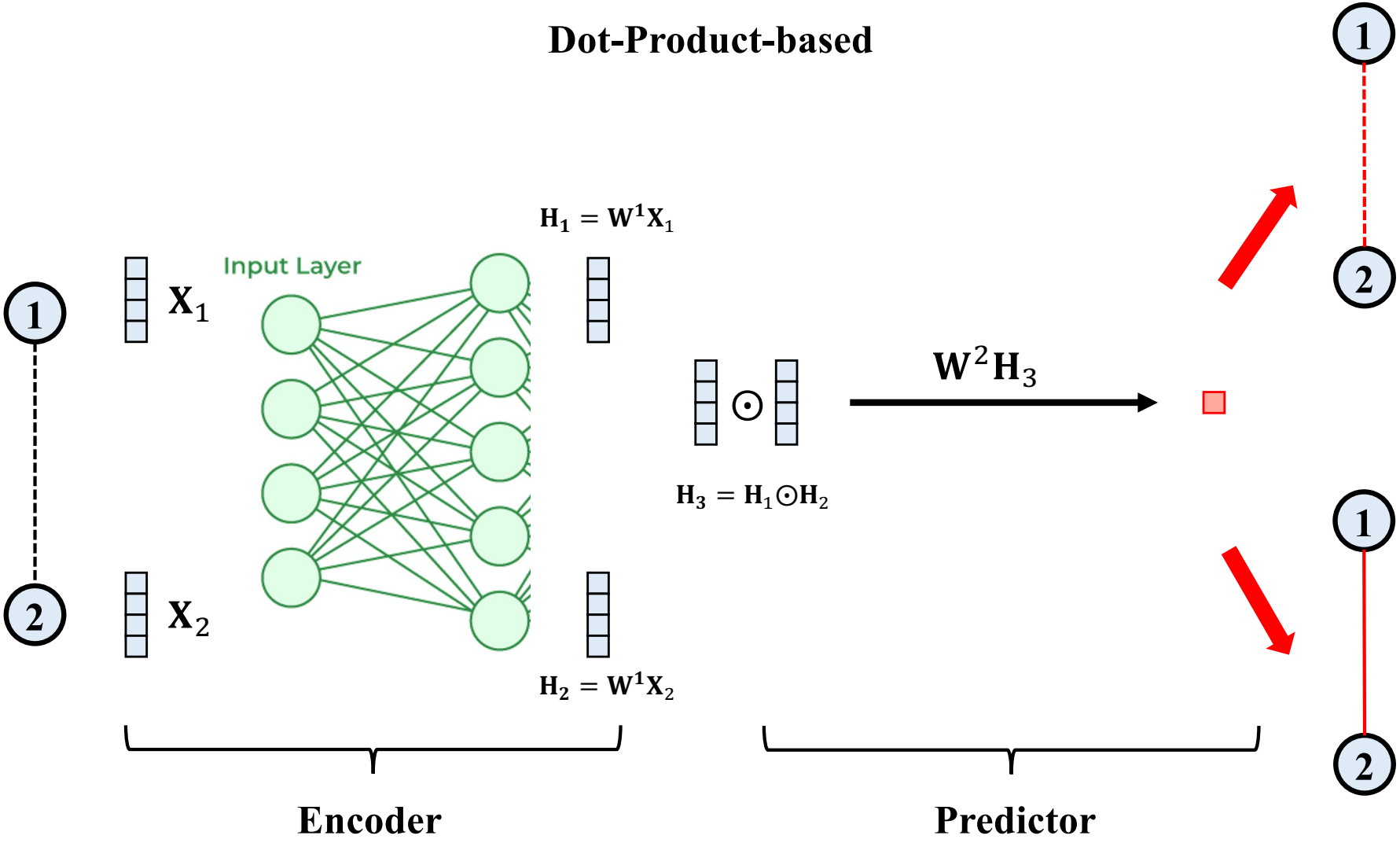
Link Prediction – MLP



Link Prediction – MLP



Link Prediction – MLP



Link Prediction – MLP

```python
class MLP_encoder(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLP_encoder, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x, edge_index):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)

        return x
```

```python
class MLP_link_predictor(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels, num_layers, dp):
        super(MLP_link_predictor, self).__init__()

        self.lins = torch.nn.ModuleList()
        self.lins.append(torch.nn.Linear(in_channels, hidden_channels))
        for _ in range(num_layers - 2):
            self.lins.append(torch.nn.Linear(hidden_channels, hidden_channels))
        self.lins.append(torch.nn.Linear(hidden_channels, out_channels))

        self.dropout = dp

    def forward(self, x):

        for lin in self.lins[:-1]:
            x = lin(x)
            x = F.relu(x)
            x = F.dropout(x, p=self.dropout, training=self.training)

        x = self.lins[-1](x)

        return x

    def score(self, x):
        for lin in self.lins[:-1]:
            x = lin(x)
            x = F.relu(x)

        x = self.lins[-1](x)

        return torch.sigmoid(x)
```

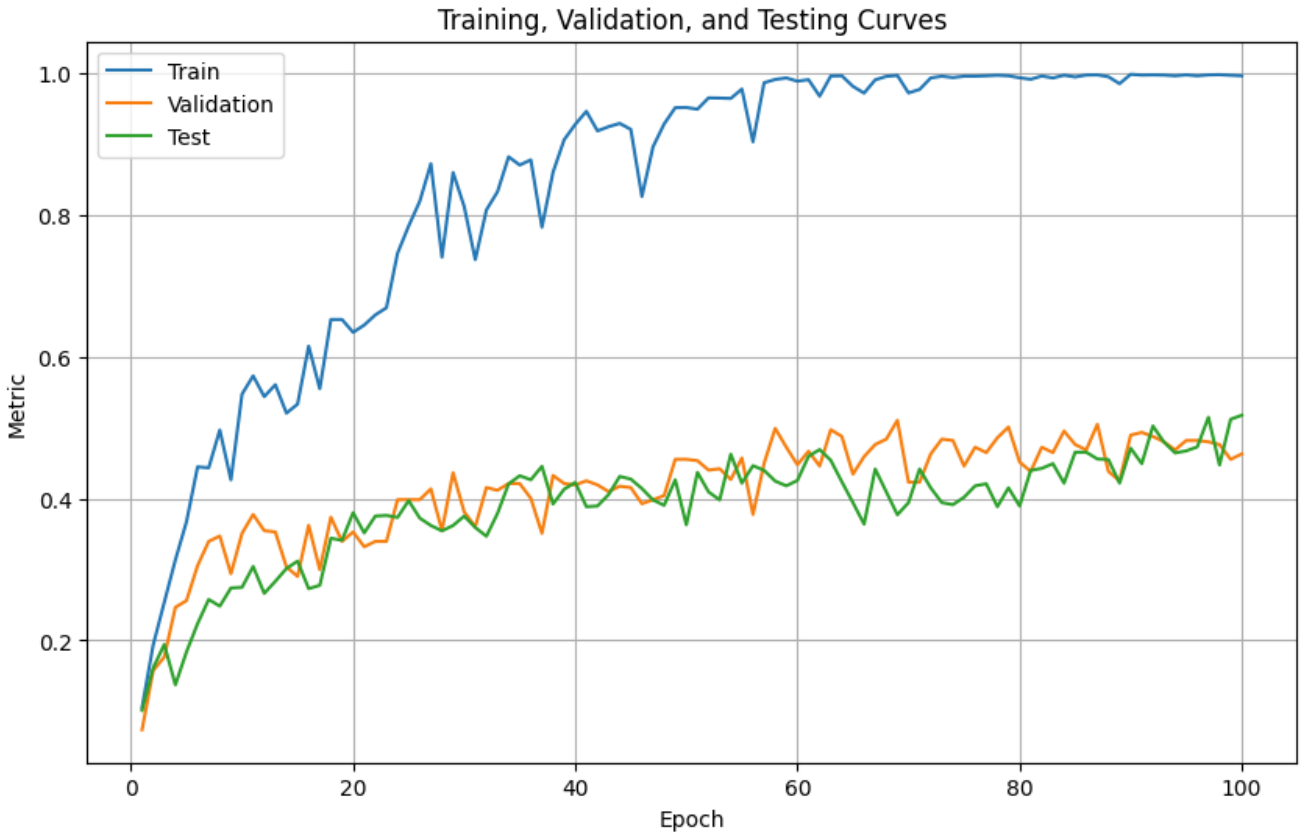
Link Prediction – MLP

```python
def train(encoder, predictor, optimizer, data, train_edge, batch_size):
    encoder.train()
    predictor.train()

    # Generate negative samples
    neg_edge = negative_sampling(data.train_edge_index, num_neg_samples=train_edge.shape[0]).t()

    # Wrap train_edge in a DataLoader
    train_dataset = TensorDataset(train_edge)
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    total_loss, count = 0, 0
    for batch in train_loader:
        batch = batch[0]  # unpack batch from DataLoader

        # Forward pass
        h = encoder(data.x, data.train_edge_index)
        pos_score = predictor(h[batch[:, 0]] * h[batch[:, 1]])
        pos_loss = -F.logsigmoid(pos_score).mean()

        neg_score = predictor(h[neg_edge[batch[:, 0]]] * h[neg_edge[batch[:, 1]]])
        neg_loss = -F.logsigmoid(-neg_score).mean()

        loss = (pos_loss + neg_loss) / 2

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Update total loss
        total_loss += loss.item() * batch.size(0)
        count += batch.size(0)

    return total_loss / count
```

$$\mathcal{L} = -\frac{1}{|\mathcal{E}^+|}\sum_{e_{ij}\in\mathcal{E}^+}\sigma(s_{ij}^+) - \frac{1}{|\mathcal{E}^-|}\sum_{e_{ij}\in\mathcal{E}^-}\sigma(-s_{ij}^-)$$

Link Prediction – Heuristic Methods

```
100%|██████████| 100/100
0.37725118483412323
```



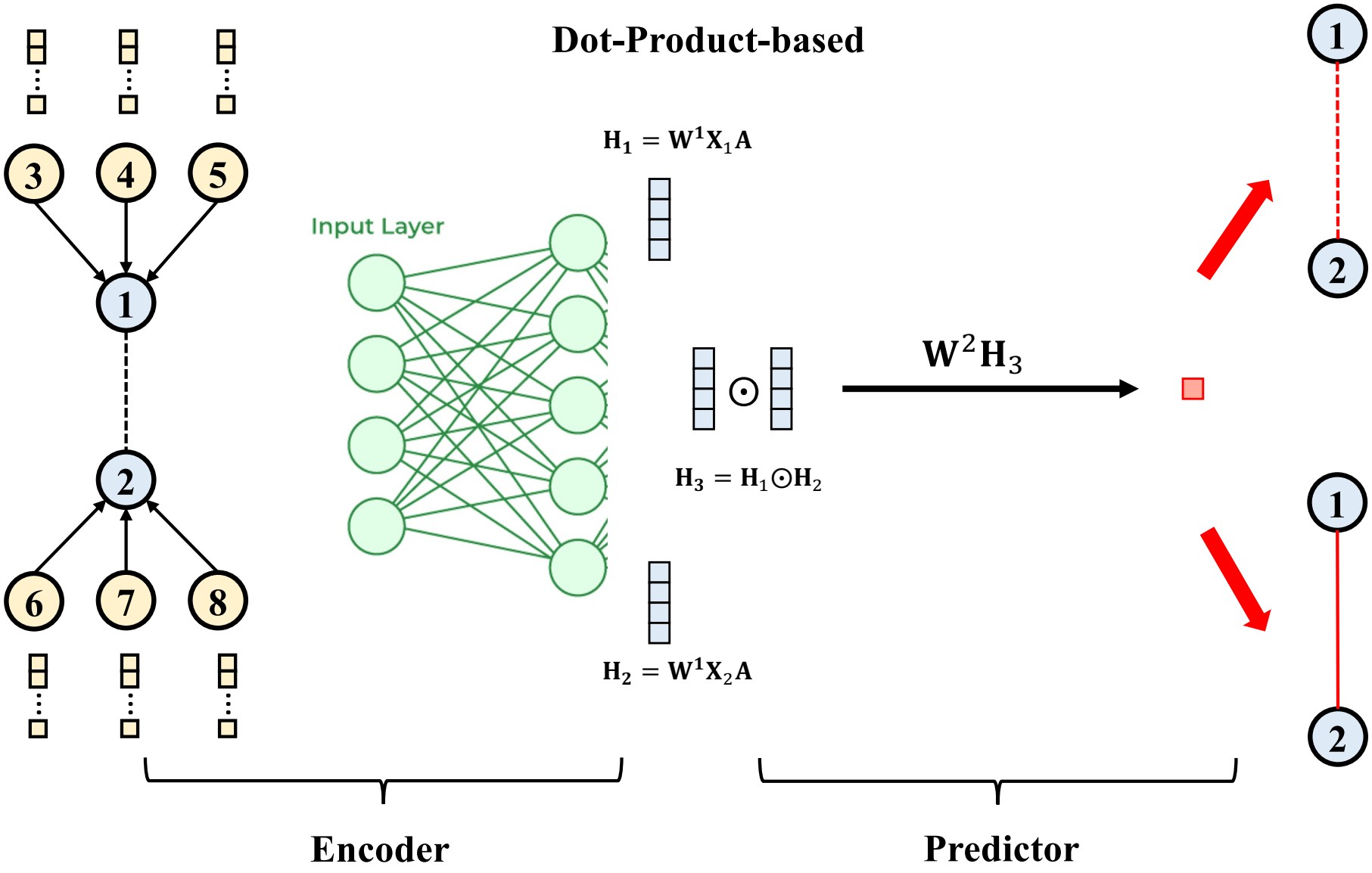
Link Prediction – GNN

1. **Heuristic Method**

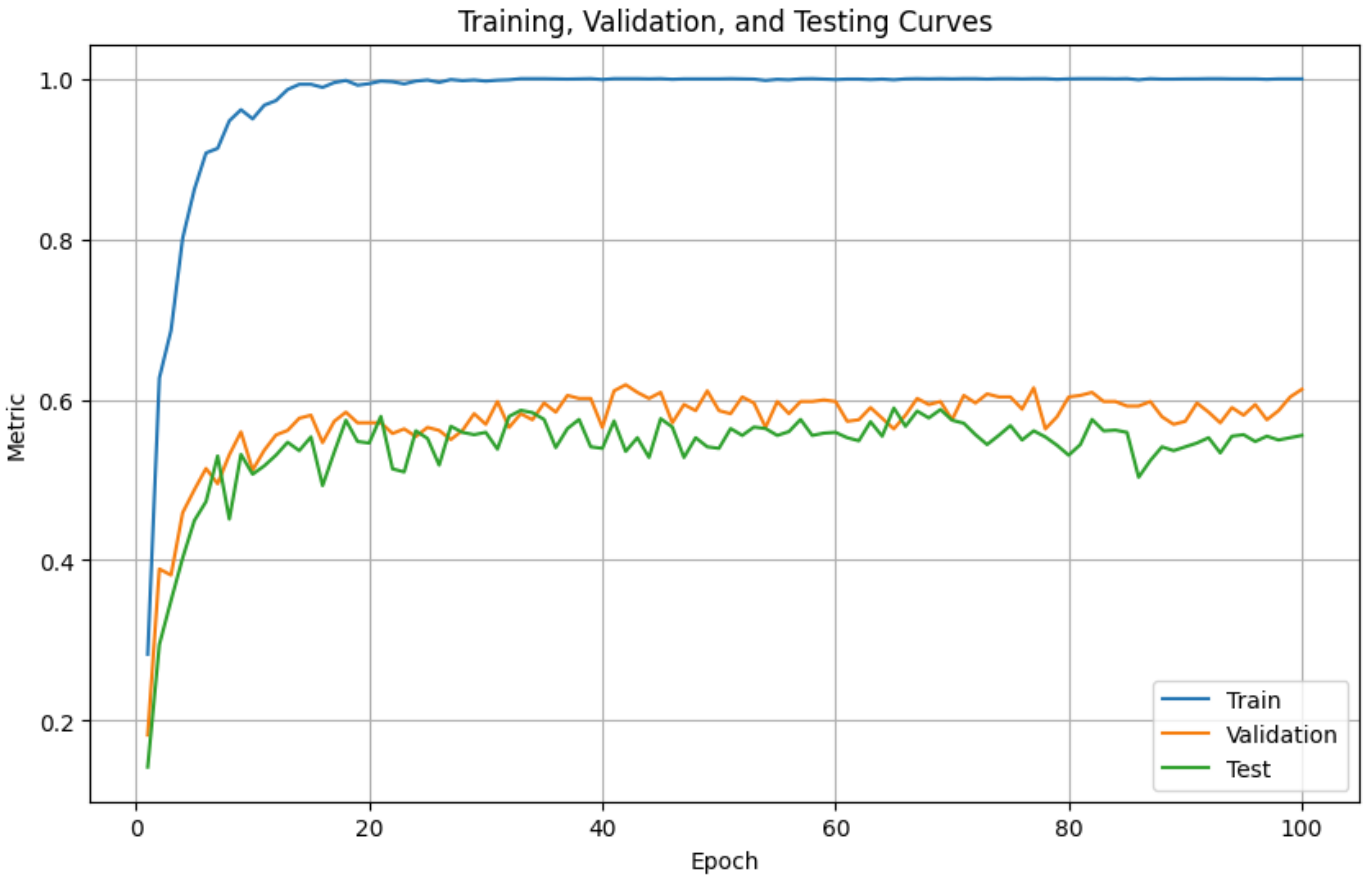
2. **MLP**

3. **Graph Neural Network**

Link Prediction – GNN



Link Prediction – GNN




```
final_test
```

```
0.5355450236966824
```

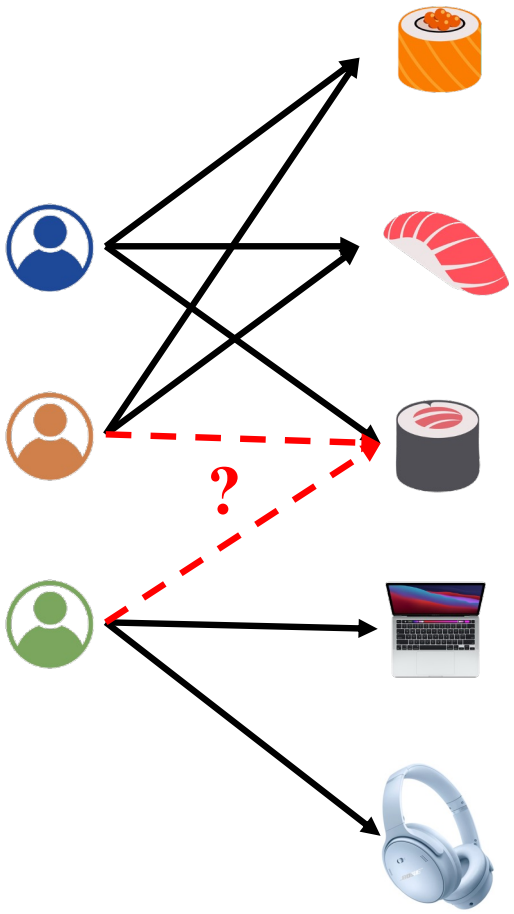
Recommender Systems

- This is essentially a form of link prediction with additional ranking
- Problem:
  - Given a set of users and items
  - We additionally have previous user behaviors
    - E.g., what they clicked on, watched, purchased, etc.
  - This can be seen as a bipartite graph of users and items
  - Thus we seek to predict what they are likely to want
    - Ranking since we prefer to present them with the most likely item they would interact with

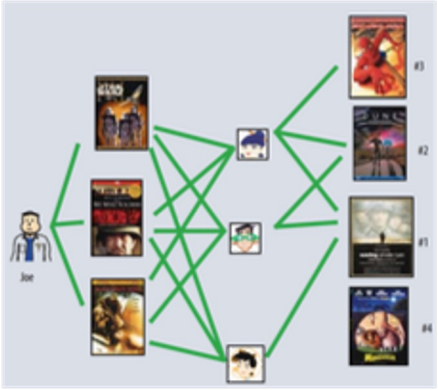
Link Prediction – Recommender Systems

1. **Heuristic Method**
2. **Matrix Factorization**
3. **Graph Neural Network**
4. **Next-item Recommendation**

Link Prediction – Recommender Systems - Heuristics



What things do you have in hand to perform recommendation?

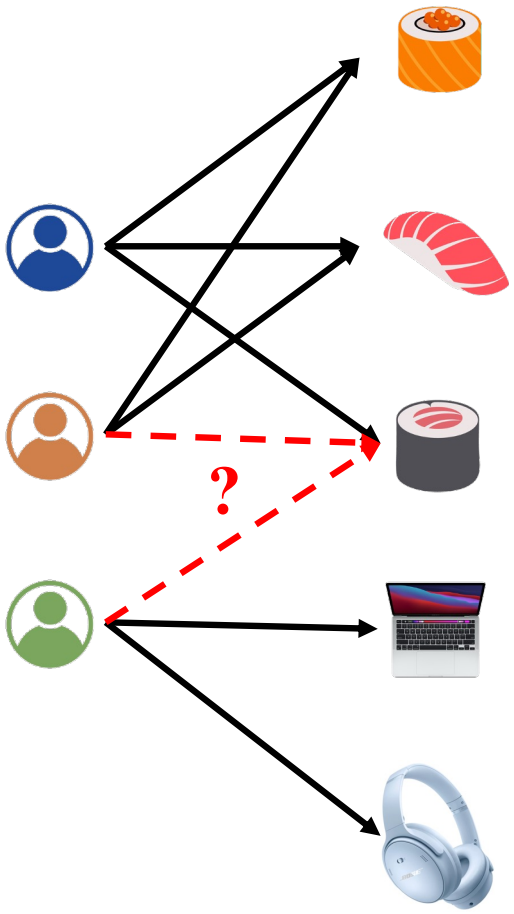


Recommender Systems

- Collaborative Filtering
  - Idea: Use the "wisdom of the crowd" to recommend
  - Assumptions:
    - We have user data in the form of interactions/ratings on items
    - Users with similar taste in past will in future

- Content Filtering
  - Idea: Create features for each item and user
    - E.g., demographics of user, genre of movie, etc.

Link Prediction – Recommender Systems - Heuristics



Item Collaborative Filtering

User Collaborative Filtering

Collaborative Filtering

	Avengers	Toy Story	Back to the Future	Harry Potter
John	5	3	4	?
Mary	5	4	5	5
Lee	2	2	4	5
Joe	3	1	1	2

The problem of collaborative filtering is to predict how well a user will like an item that he/she has not rated given a set of historical preference judgments for a community of users.

Item-based

- **Item-based Nearest Neighbor**
  - Given a user *u*, generate a prediction for item *i* by using the weighted sum of *u*'s ratings for items that are most similar to *i*.

	Avengers	Toy Story	Back to the Future	Harry Potter
John	5	3	4	?
Mary	5	4	5	5
Lee	2	2	4	5
Joe	3	1	1	2

$$pred(u,i) = \frac{\sum_{j \in ratedItems(u)} sim(i,j) \cdot r_{uj}}{\sum_{j \in ratedItems(u)} sim(i,j)}$$

User-based

- **User-based Nearest Neighbor**
  - Given a user *u*, generate a prediction for an item *i* by using the ratings for *i* from other users in *u*'s neighborhood

	Avengers	Toy Story	Back to the Future	Harry Potter
John	5	3	4	?
Mary	5	4	5	5
Lee	2	2	4	5
Joe	3	1	1	2

Average ratings of user u

Average ratings of neighbor n

$$pred(u,i) = \bar{r}_u + \frac{\sum_{n \in neighbors(u)} sim(u,n) \cdot (r_{ni} - \bar{r}_n)}{\sum_{n \in neighbors(u)} sim(u,n)}$$

Link Prediction – Recommender Systems - MF

1. Heuristic Method
2. Matrix Factorization
3. Graph Neural Network
4. Next-item Recommendation

Netflix Prize

- Challenge: Improve their current system by 10%
- Prize: $1 million
- Started Oct. 2006 and lasted ~3 years

- Dataset:
  - ~100m ratings from ~500k users for ~20k movies

Link Prediction – Recommender Systems - MF

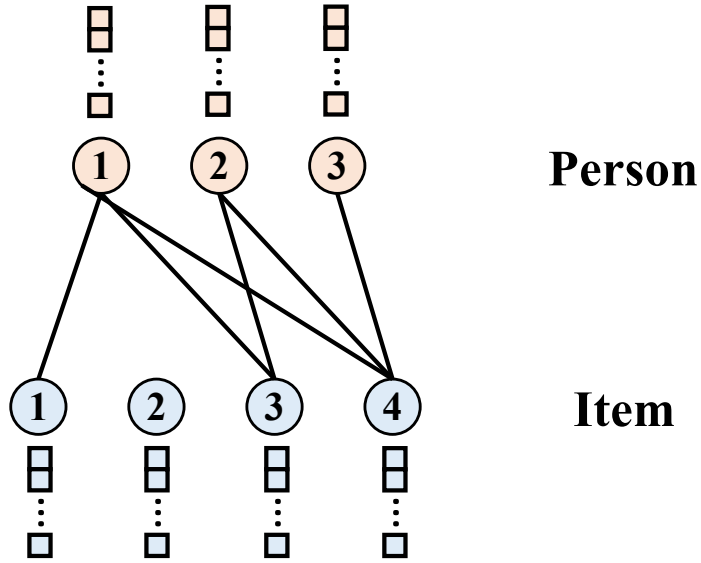
Netflix Prize

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(pred_i - real_i)^2}$$

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell				

Link Prediction – Recommender Systems - MF

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4\times 3}$$



$$\mathbf{M} = \mathbf{U}\mathbf{I}^{\mathrm{T}}$$

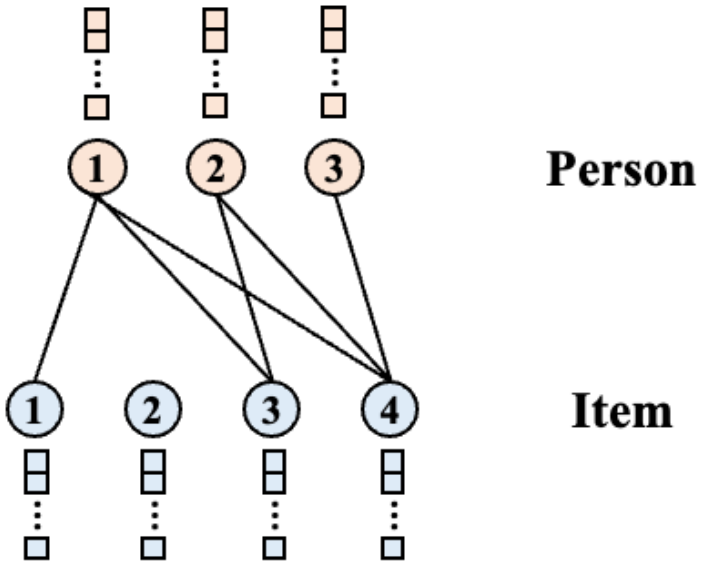
$\mathbf{U}$: $\mathbb{R}^{4\times 16}$

$\mathbf{I}$: $\mathbb{R}^{3\times 16}$

What kind of user/item embeddings would be most predictive of user-item interaction?

Link Prediction – Recommender Systems - MF

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4\times 3}$$



$$\mathbf{M}' = \mathbf{U}\mathbf{I}^{\mathrm{T}}$$

$\mathbf{U}$: $\mathbb{R}^{4\times 16}$

$\mathbf{I}$: $\mathbb{R}^{3\times 16}$

$$\min_{\mathbf{U},\mathbf{I}} ||\mathbf{M} - \mathbf{M}'||_2^2 + ||\mathbf{U}||_2^2 + ||\mathbf{I}||_2^2$$

What kind of user/item embeddings would be most predictive of user-item interaction?

Link Prediction – Recommender Systems - MF

$$\min_{\mathbf{U},\mathbf{I}} \mathcal{L} = \left|\left|\mathbf{M} - \mathbf{M}'\right|\right|_2^2 + \left|\left|\mathbf{U}\right|\right|_2^2 + \left|\left|\mathbf{I}\right|\right|_2^2$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\mathbf{M}_{ij} - \mathbf{M}'_{ij}\right)^2 + \sum_{i=1}^{n}\sum_{j=1}^{k}\left(\mathbf{U}_{ij} - \mathbf{U}'_{ij}\right)^2 + \cdots$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\mathbf{M}_{ij} - \sum_{k=1}^{K}\mathbf{U}_{ik}\mathbf{I}_{\mathrm{jk}}\right)^2 + \sum_{i=1}^{n}\sum_{j=1}^{k}\left(\mathbf{U}_{ij} - \mathbf{U}'_{ij}\right)^2 + \cdots$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4\times 3}$$

$$\mathbf{M}' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & \boxed{1} & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4\times 3}$$

$$\mathcal{L}_{ab} = \left(\mathbf{M}_{ab} - \sum_{k=1}^{K}\mathbf{U}_{ak}\mathbf{I}_{\mathrm{bk}}\right)^2$$

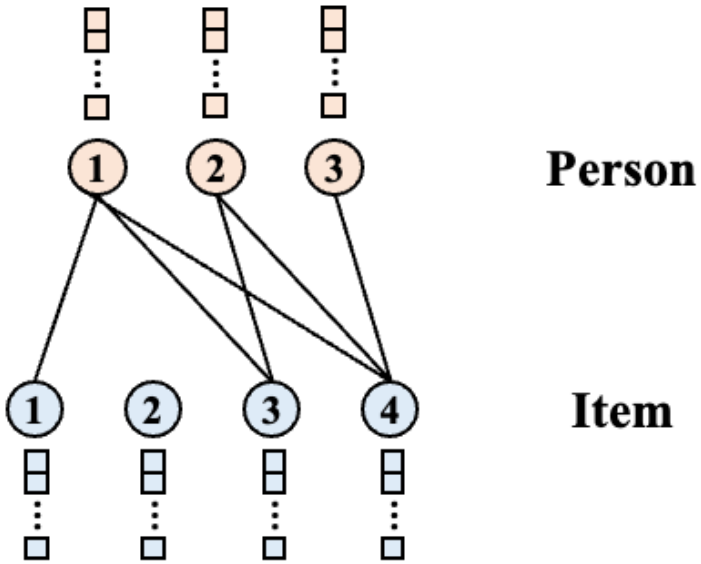
$$\frac{\partial \mathcal{L}_{ab}}{\partial \mathbf{U}_{ap}} = -2(\mathbf{M}_{ab} - \sum_{k=1}^{K}\mathbf{U}_{ak}\mathbf{I}_{\mathrm{bk}})\mathbf{I}_{bp}$$

$$= -\mathbf{2}(\mathbf{M}_{ab} - \mathbf{M}'_{ab})\mathbf{I}_{bp}$$

$\mathbf{U}_3$ ⊙ $\mathbf{I}_2$

Link Prediction – Recommender Systems - MF

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4\times 3}$$



$$\mathbf{M}' = \mathbf{U}\mathbf{I}^{\mathrm{T}}$$

$\mathbf{U}$: $\mathbb{R}^{4\times 16}$

$\mathbf{I}$: $\mathbb{R}^{3\times 16}$

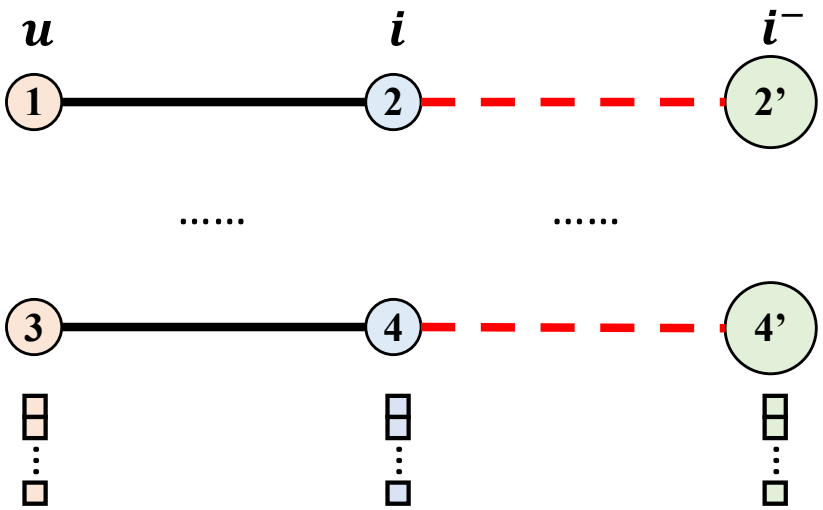
$$\min_{\mathbf{U},\mathbf{I}} ||\mathbf{M} - \mathbf{M}'||_2^2 + ||\mathbf{U}||_2^2 + ||\mathbf{I}||_2^2$$

$$\mathcal{L}_{\mathrm{BPR}} = \sum_{(u,i,i^-)\in O} -\ln \sigma(y_{ui} - y_{ui^-}),$$

What kind of user/item embeddings would be most predictive of user-item interaction?

Link Prediction – Recommender Systems - MF

$$\mathcal{L}_{\text{BPR}} = \sum_{(u,i,i^-)\in O} -\ln \sigma(y_{ui} - y_{ui^-}),$$



$$y_{ui} = \mathbf{e}_u^\top \mathbf{e}_i \text{ and } y_{ui^-} = \mathbf{e}_u^\top \mathbf{e}_{i^-}$$

Link Prediction – Recommender Systems - MF

```
for i, batch in enumerate(dataloader):
    batch = batch_to_gpu(batch, args.device)

    user_embs, pos_item_embs, neg_item_embs, user_embs0, pos_item_embs0, neg_item_embs0 = model(
        batch)

    bpr_loss = cal_bpr_loss(
        batch['users'], user_embs, pos_item_embs, neg_item_embs)

    # l2 regularization
    l2_loss = cal_l2_loss(
        user_embs0, pos_item_embs0, neg_item_embs0, user_embs0.shape[0])

    batch_loss = bpr_loss + args.l2 * l2_loss

    optimizer.zero_grad()
    batch_loss.backward()
    optimizer.step()
```

Link Prediction – Recommender Systems - MF

```python
class MF(nn.Module):
    def __init__(self, args):
        super(MF, self).__init__()
        self.args = args

        self._init_weight()

    def _init_weight(self):
        initializer = nn.init.xavier_uniform_
        self.embeds = nn.Parameter(initializer(torch.empty(
            self.args.n_users + self.args.n_items, self.args.embedding_dim)))

    def generate(self):
        user_embs, item_embs = self.embeds[:self.args.n_users,
                                           :], self.embeds[self.args.n_users:, :]

        return user_embs, item_embs

    def batch_generate(self, user, pos_item, neg_item):
        user_embs = self.embeds[user]
        pos_item_embs = self.embeds[pos_item]
        neg_item_embs = self.embeds[neg_item]

        return user_embs, pos_item_embs, neg_item_embs

    def forward(self, batch):
        user = batch['users']
        pos_item = batch['pos_items']
        neg_item = batch['neg_items']  # [batch_size, n_negs * K]

        batch_size = user.shape[0]

        user_embs, pos_item_embs, neg_item_embs = self.batch_generate(
            user, pos_item, neg_item)

        return user_embs, pos_item_embs, neg_item_embs, user_embs, pos_item_embs, neg_item_embs
```

Link Prediction – Recommender Systems - MF

Bob

Ground-truth	Recommended@5
1	2
5	7
6	5
	9
	8

$$\text{Recall} = \frac{\text{Relevant Item} \cap \text{Recommender Item}}{\text{Relevant Item}}$$

$$\text{Precision} = \frac{\text{Relevant Item} \cap \text{Recommender Item}}{\text{Recommender Item}}$$

$$CG(k) = \sum_{i=1}^{k} G_i \qquad DCG(k) = \sum_{i=1}^{k} \frac{G_i}{log_2(i+1)}$$

$$IDCG(k) = \sum_{i=1}^{|I(k)|} \frac{G_i}{log_2(i+1)} \qquad NDCG(k) = \frac{DCG(k)}{IDCG(k)}$$

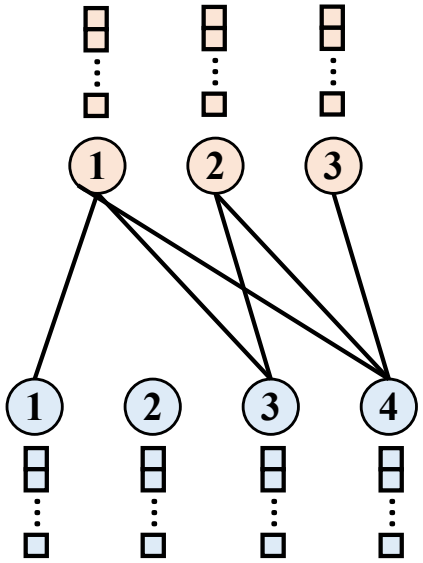
Link Prediction – Recommender Systems - MF

```
=====MF=====
cpu
reading train and test user-item set ...
loading over ...
# of users: 29858
# of items: 40988
# of edges: 513731
# of training edges: 393601
# of validation edges: 58785
# of testing edges: 61345
Network density: 0.00041977670266371195
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+--
|   Time  | Epoch |                     Recall                    |                     NDCG                      |                   Precision                   |
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+--
| 66.6866 |   0   | [0.0094 0.0142 0.0229 0.0339 0.0557 0.081 ] | [0.0128 0.0142 0.017  0.0205 0.0263 0.0321] | [0.0108 0.0098 0.0078 0.0058 0.0039 0.0028] | [
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+--
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+-
|   Time  | Epoch |                     Recall                    |                     NDCG                      |                   Precision                   |
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+-
| 387.1903 |   5   | [0.0267 0.0361 0.0556 0.0806 0.1325 0.1887] | [0.0312 0.0337 0.0403 0.048  0.061  0.0729] | [0.0224 0.0183 0.0141 0.0105 0.0071 0.0051] |
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+-
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+-
|   Time  | Epoch |                     Recall                    |                     NDCG                      |                   Precision                   |
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+-
| 725.1866 |   10  | [0.0292 0.0399 0.061  0.0904 0.1511 0.2158] | [0.0328 0.0361 0.0433 0.0523 0.0675 0.0811] | [0.0235 0.0195 0.015  0.0115 0.0078 0.0057] |
+---------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+-
+----------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
|   Time   | Epoch |                     Recall                    |                     NDCG                      |                   Precision                   |
+----------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
| 1046.2437 |   15  | [0.0313 0.042  0.0639 0.0955 0.1599 0.2296] | [0.0342 0.0375 0.045  0.0547 0.0707 0.0853] | [0.0246 0.0201 0.0155 0.0119 0.0082 0.006 ] |
+----------+-------+-----------------------------------------------+-----------------------------------------------+-----------------------------------------------+
```

Link Prediction – Recommender Systems - GNN

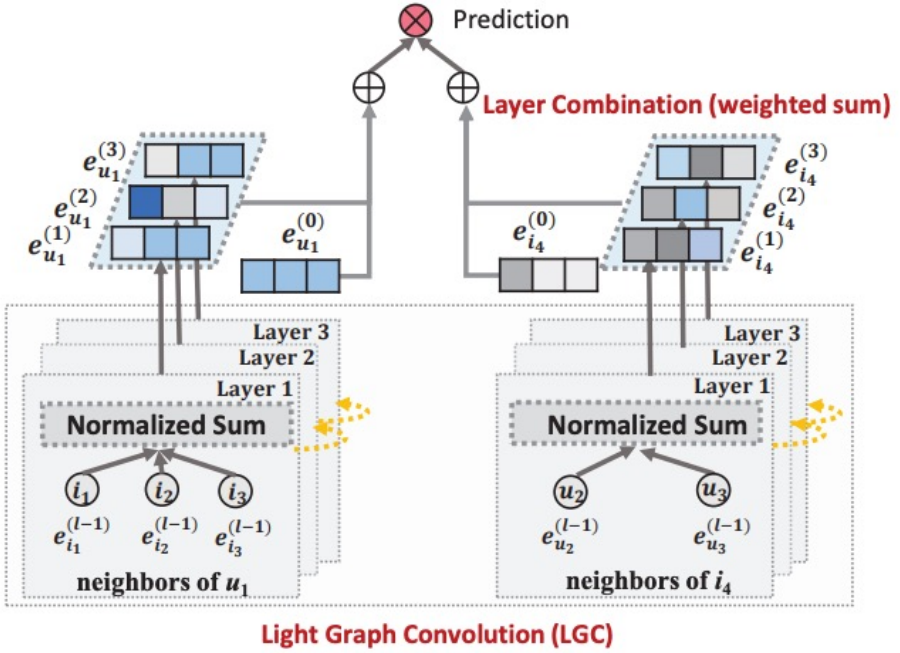
1. Heuristic Method
2. Matrix Factorization
3. Graph Neural Network
4. Next-item Recommendation

Link Prediction – Recommender Systems - GNN



Person

Item



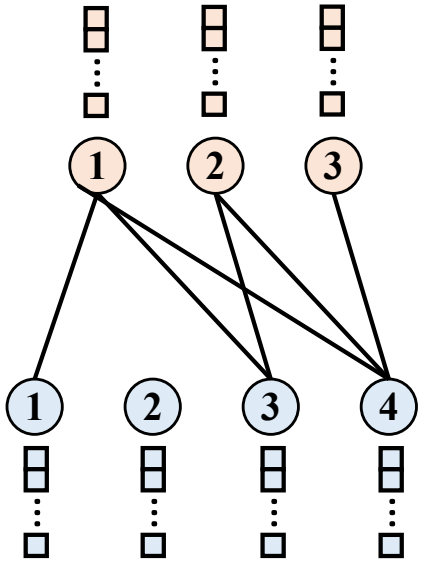
$$\mathbf{e}_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)},$$

$$\mathbf{e}_i^{(k+1)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} \mathbf{e}_u^{(k)}.$$

$$y_{ui} = \mathbf{e}_u^\top \mathbf{e}_i \text{ and } y_{ui^-} = \mathbf{e}_u^\top \mathbf{e}_{i^-}$$

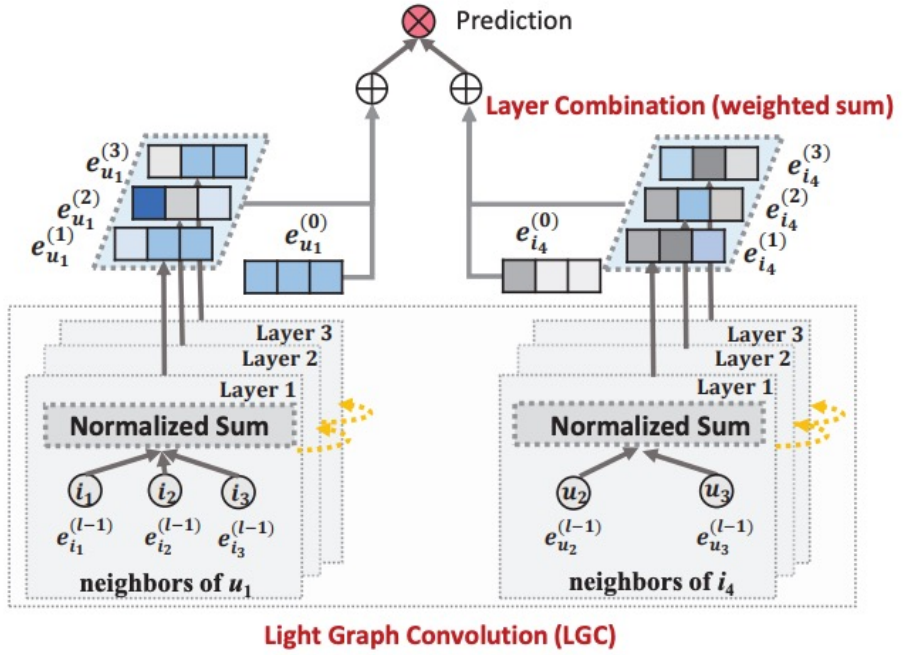
$$\mathcal{L}_{\text{BPR}} = \sum_{(u,i,i^-) \in O} -\ln \sigma(y_{ui} - y_{ui^-}),$$

Link Prediction – Recommender Systems - GNN



Person

Item



$$\mathbf{e}_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)},$$

$$\mathbf{e}_i^{(k+1)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} \mathbf{e}_u^{(k)}.$$

$$\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}; \quad \mathbf{e}_i = \sum_{k=0}^{K} \alpha_k \mathbf{e}_i^{(k)},$$

$$y_{ui} = \mathbf{e}_u^\top \mathbf{e}_i \text{ and } y_{ui^-} = \mathbf{e}_u^\top \mathbf{e}_{i^-}$$

$$\mathcal{L}_{\text{BPR}} = \sum_{(u,i,i^-) \in O} -\ln \sigma(y_{ui} - y_{ui^-}),$$

Link Prediction – Recommender Systems - GNN

```python
def forward(self, batch=None):
    user = batch['users']
    pos_item = batch['pos_items']
    neg_item = batch['neg_items']

    user_embs, pos_item_embs, neg_item_embs = self.batch_generate(
        user, pos_item, neg_item)

    return user_embs, pos_item_embs, neg_item_embs, self.embeds[user], self.embeds[pos_item], self.embeds[neg_item]

def pooling(self, embeddings):
    # [-1, n_hops, channel]
    if self.args.aggr == 'mean':
        return embeddings.mean(dim=1)
    elif self.args.aggr == 'sum':
        return embeddings.sum(dim=1)
    elif self.args.aggr == 'concat':
        return embeddings.view(embeddings.shape[0], -1)
    else:  # final
        return embeddings[:, -1, :]

def generate(self):
    user_gcn_embs, item_gcn_embs = self.gcn(
        self.embeds, self.adj_sp_norm, self.edge_index, self.edge_weight, self.deg)

    user_embs, item_embs = self.pooling(
        user_gcn_embs), self.pooling(item_gcn_embs)

    return user_embs, item_embs

def generate_layers(self):
    return self.gcn(self.embeds, self.adj_sp_norm, self.edge_index, self.edge_weight, self.deg)
```

Link Prediction – Recommender Systems - GNN

Model	Metric	MF	NGCF	LightGCN
Gowalla	Recall@20	13.90	13.98	17.24
	NDCG@20	9.15	9.08	12.13
Yelp	Recall@20	5.59	5.58	6.88
	NDCG@20	2.8	2.83	3.59
Amazon	Recall@20	6.58	6.58	6.96
	NDCG@20	3.31	3.36	3.54

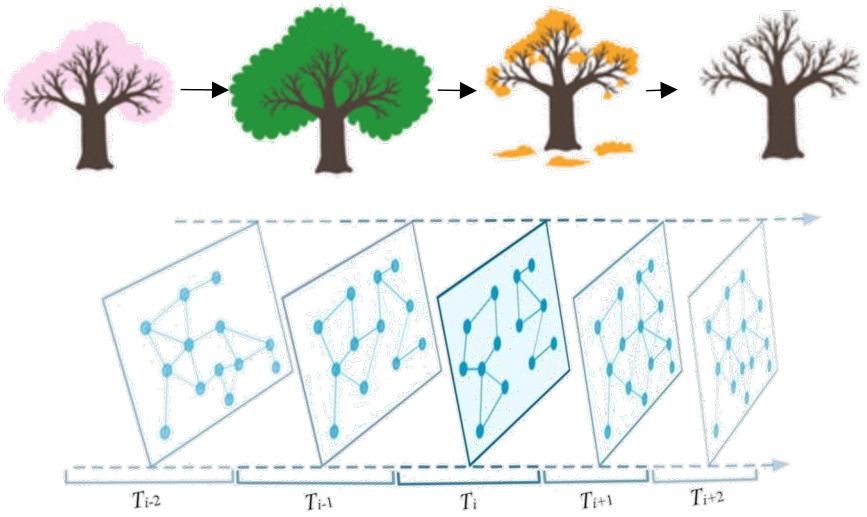
https://github.com/YuWVandy/CAGCN

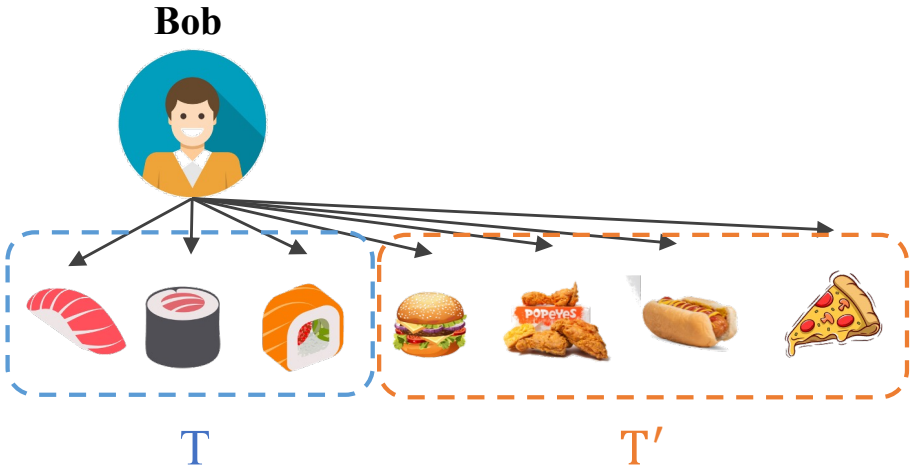
Link Prediction – Recommender Systems – Next Item

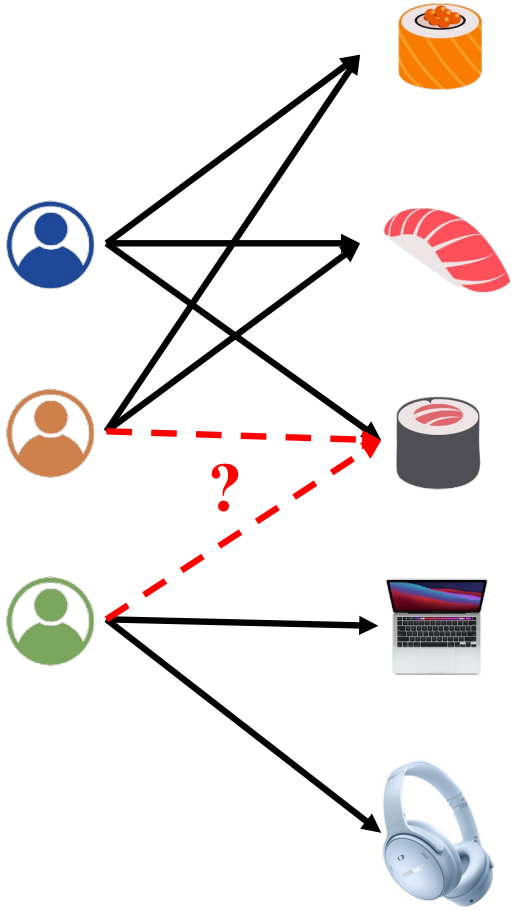
1. Heuristic Method
2. Matrix Factorization
3. Graph Neural Network
4. Next-item Recommendation

Link Prediction – Recommender Systems – Next Item

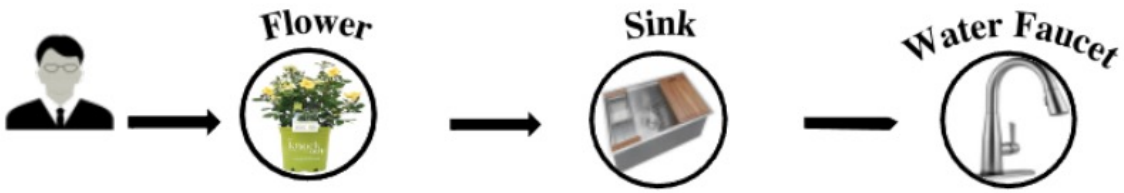
Distribution Shift Issue





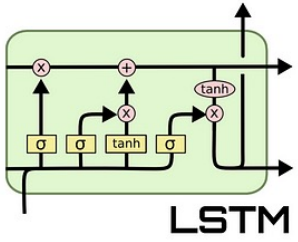


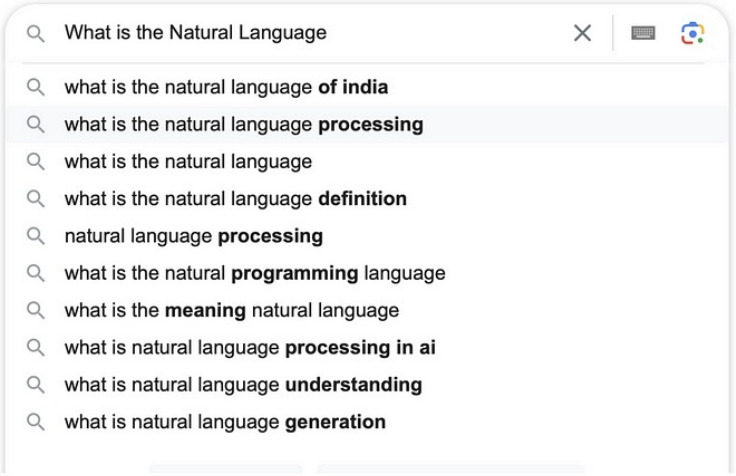
Link Prediction – Recommender Systems – Next Item



TensorFlow

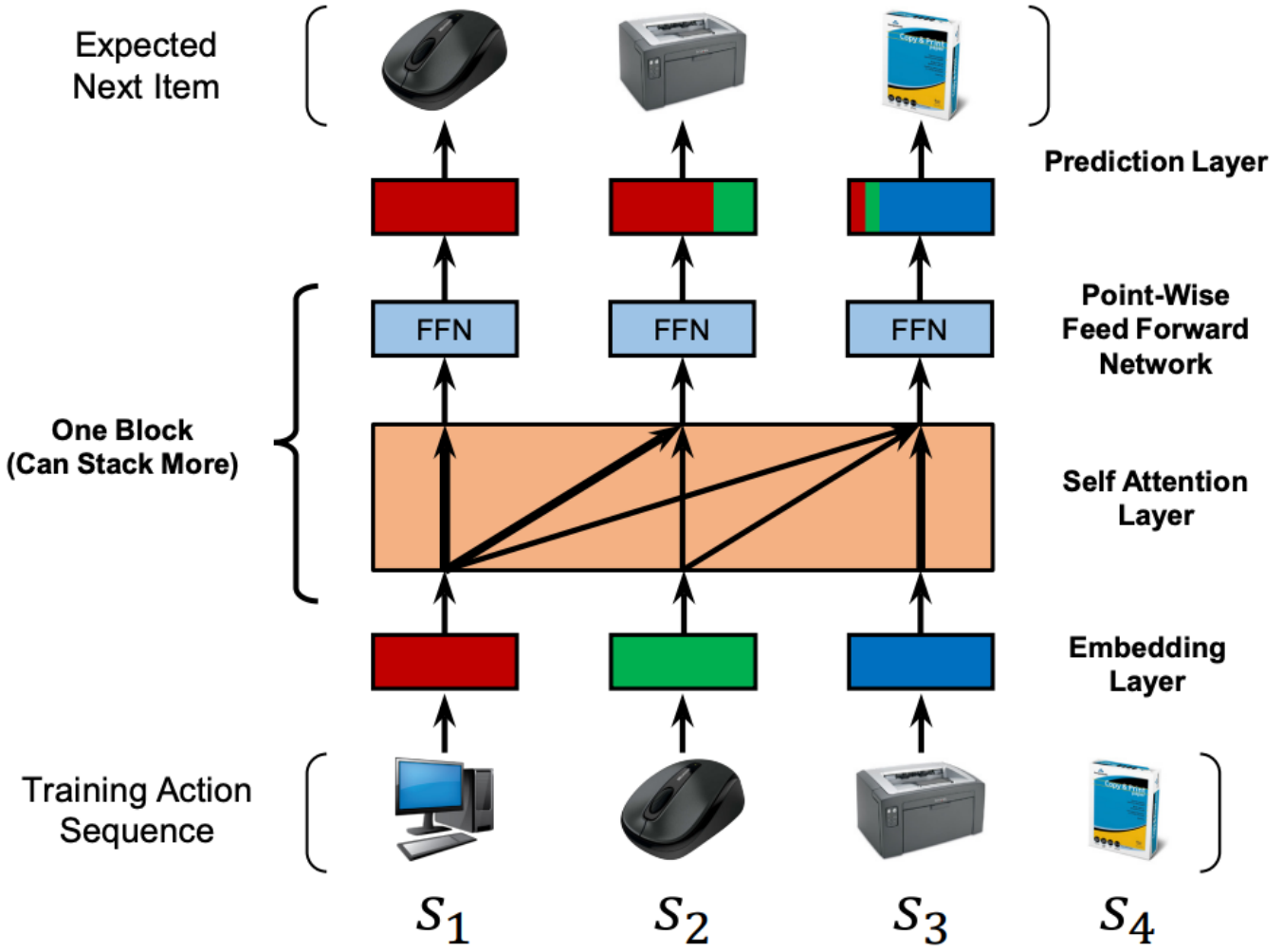
Google





Next Word Prediction

Link Prediction – Recommender Systems – Next Item



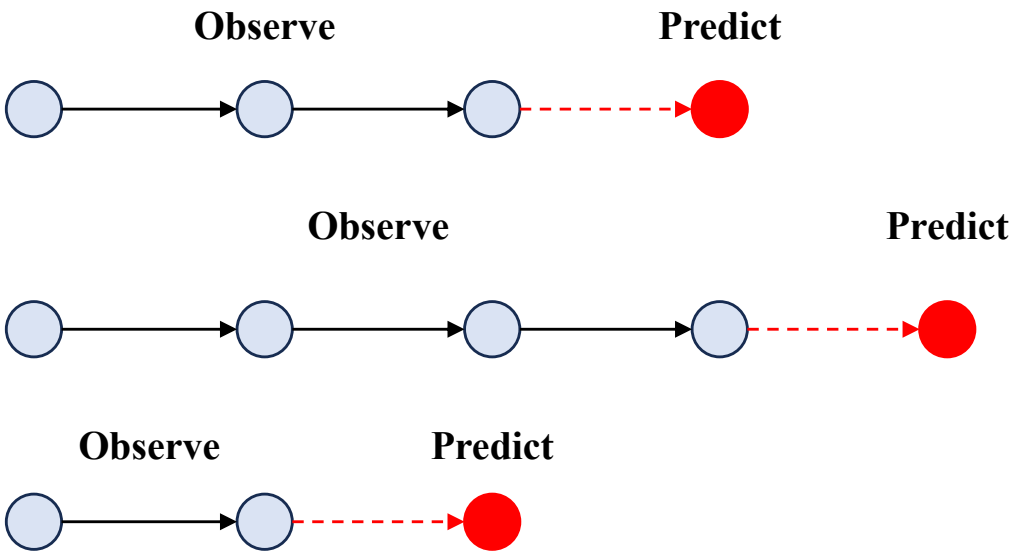
RNN

LSTM

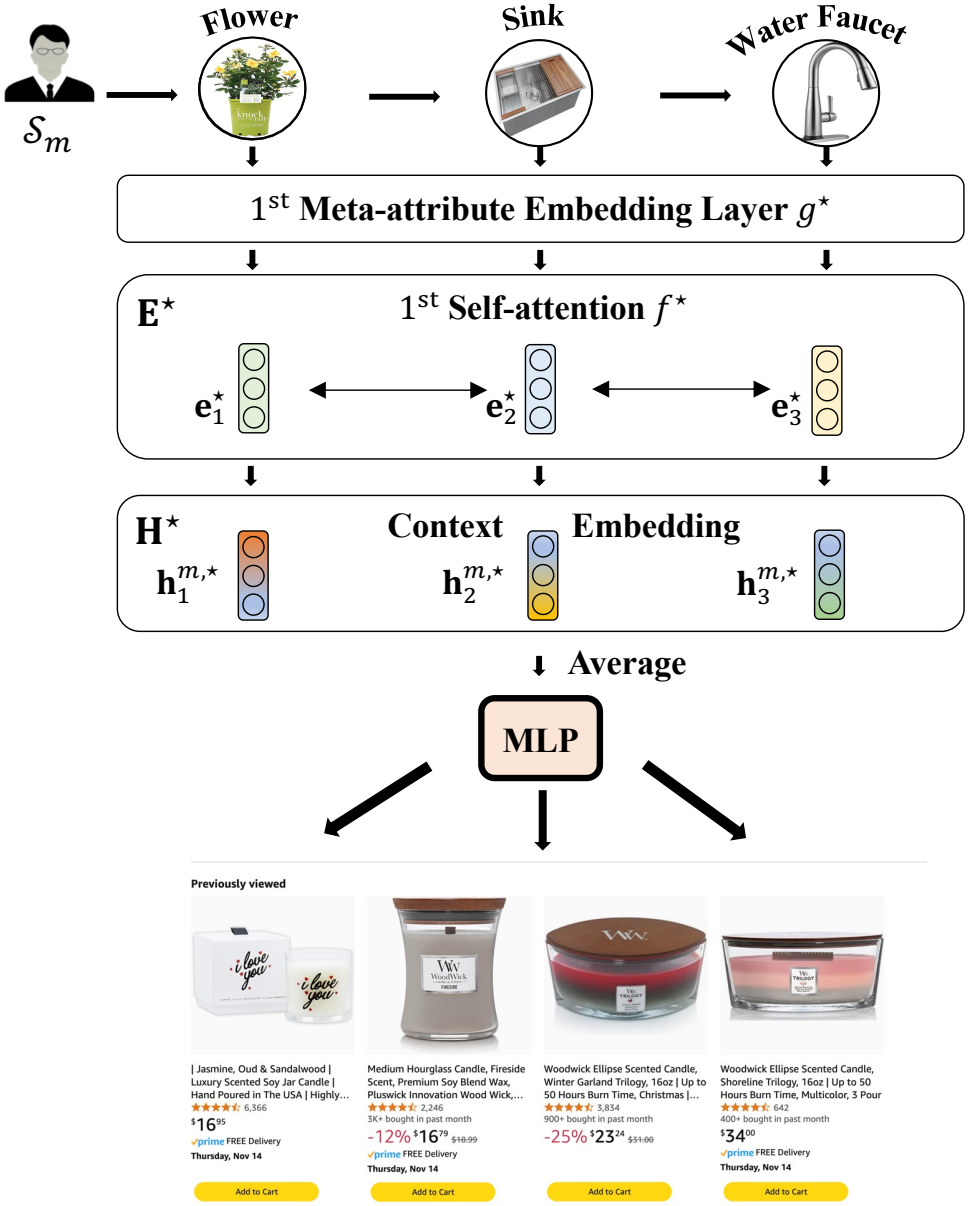
Transformer

Link Prediction – Recommender Systems – Next Item

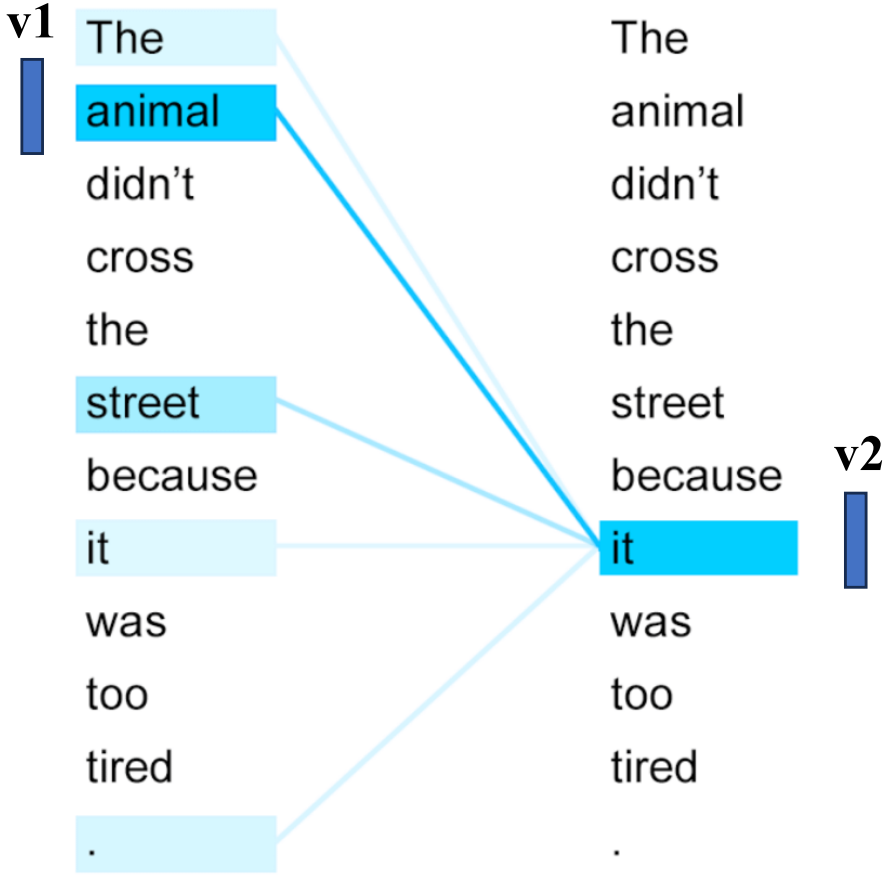
$$\prod_{k \in K} P(\text{Item}_i^k | \text{Item}_1^k, \text{Item}_2^k, \ldots, \text{Item}_{i-1}^k)$$



Link Prediction – Recommender Systems – Next Item



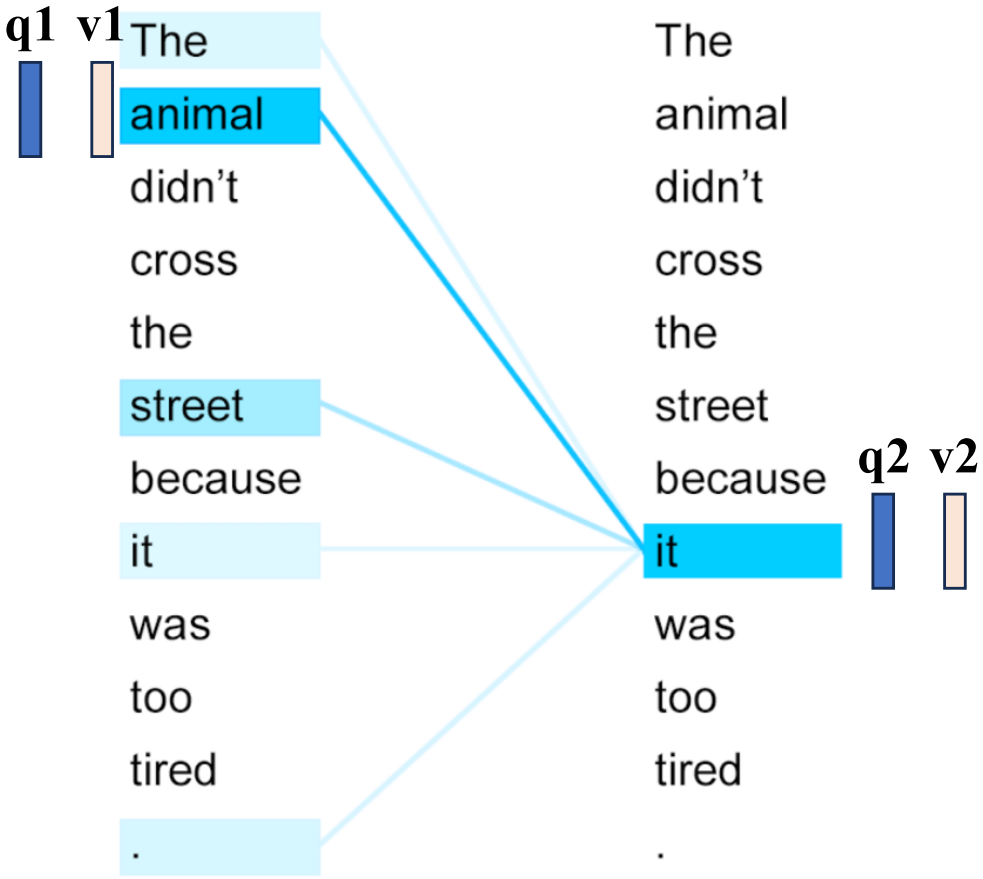
Link Prediction – Recommender Systems – Next Item



$$\text{similarity} = v_1^T v_2 = v_2^T v_1$$

My attention to you != Your attention to me

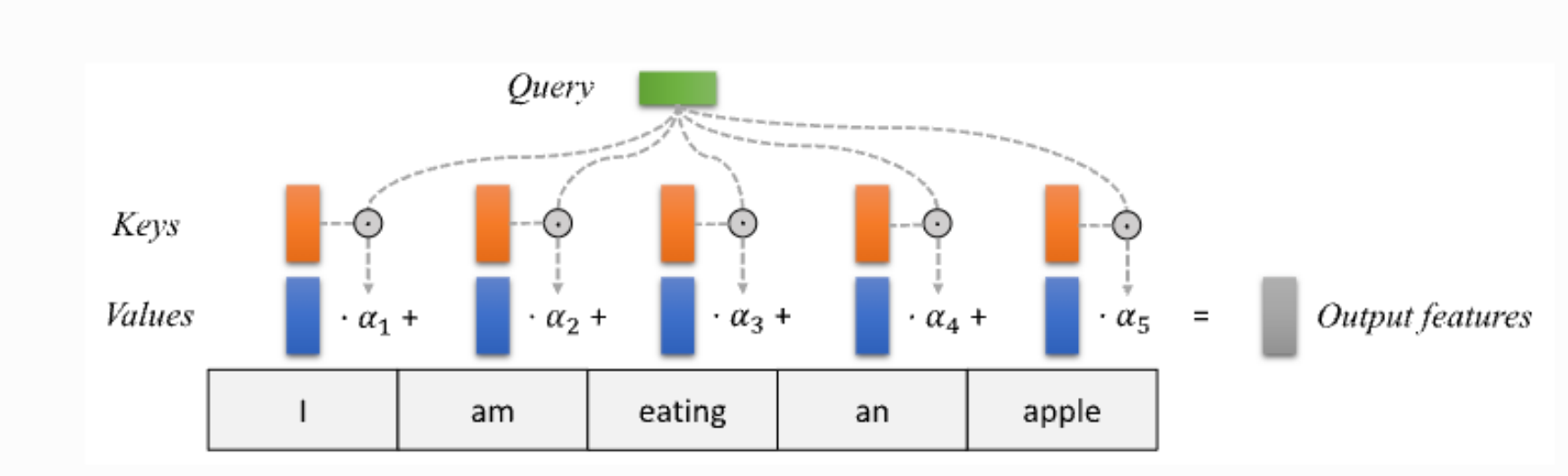
Link Prediction – Recommender Systems – Next Item



My attention to you: My query to your key

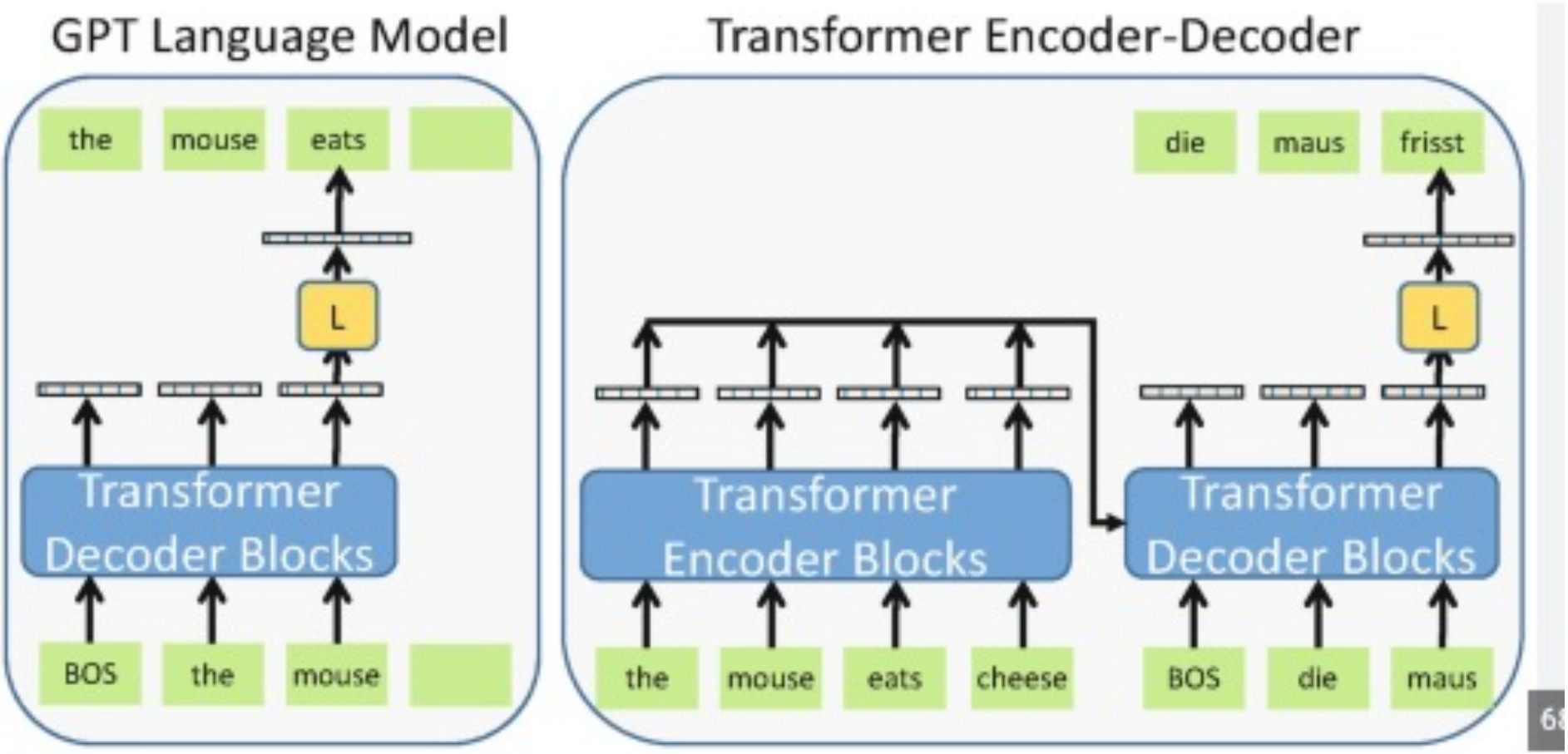
Your attention to me: Your query to my key

Link Prediction – Recommender Systems – Next Item



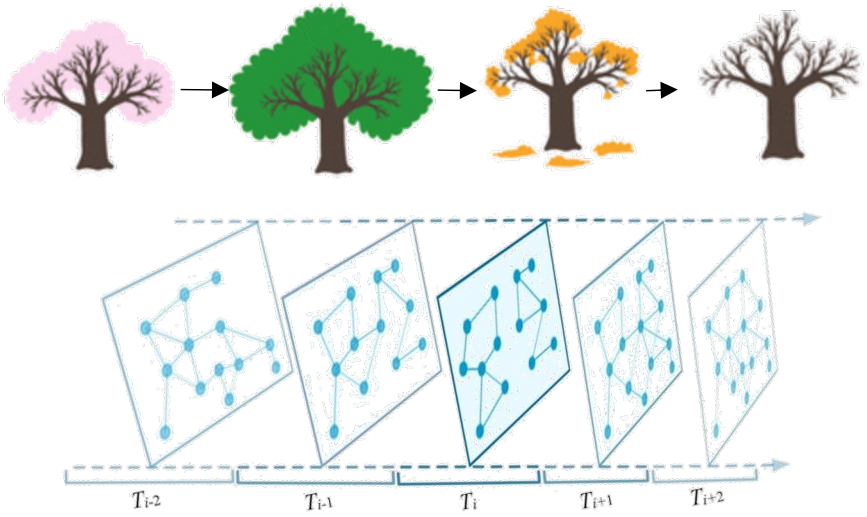
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

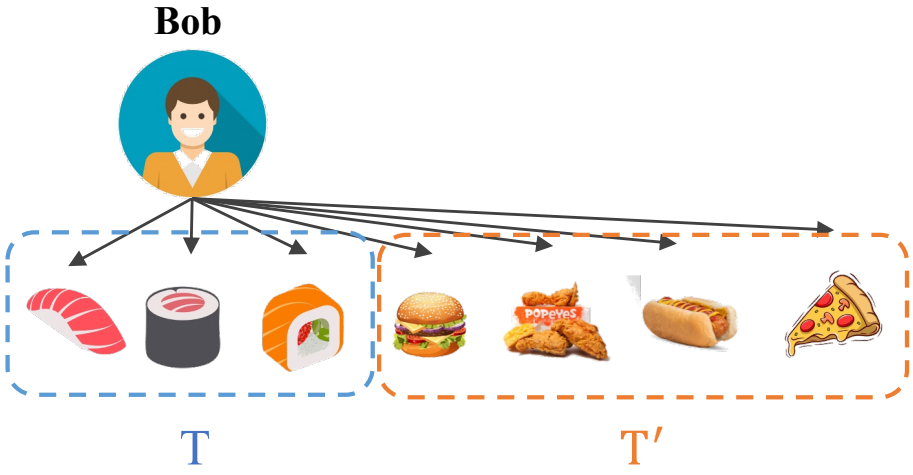
Link Prediction – Recommender Systems – Next Item

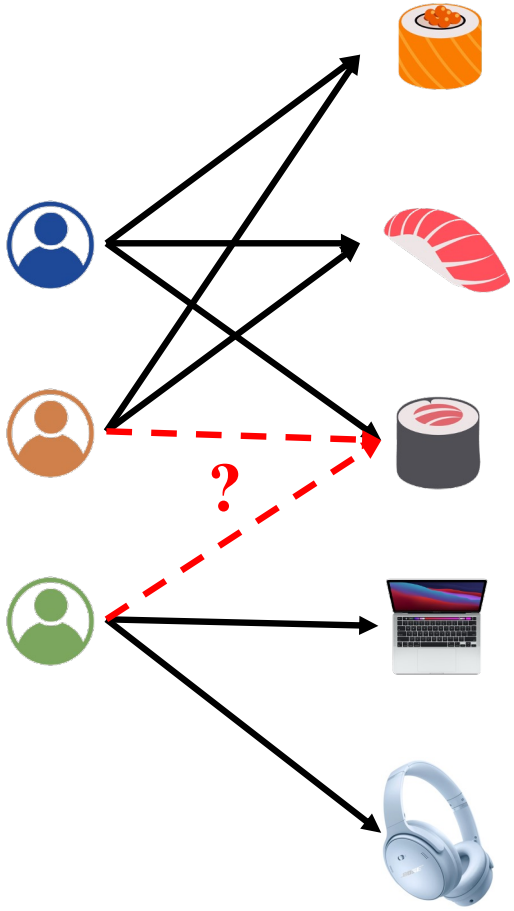


Link Prediction – Recommender Systems – Next Item

Distribution Shift Issue

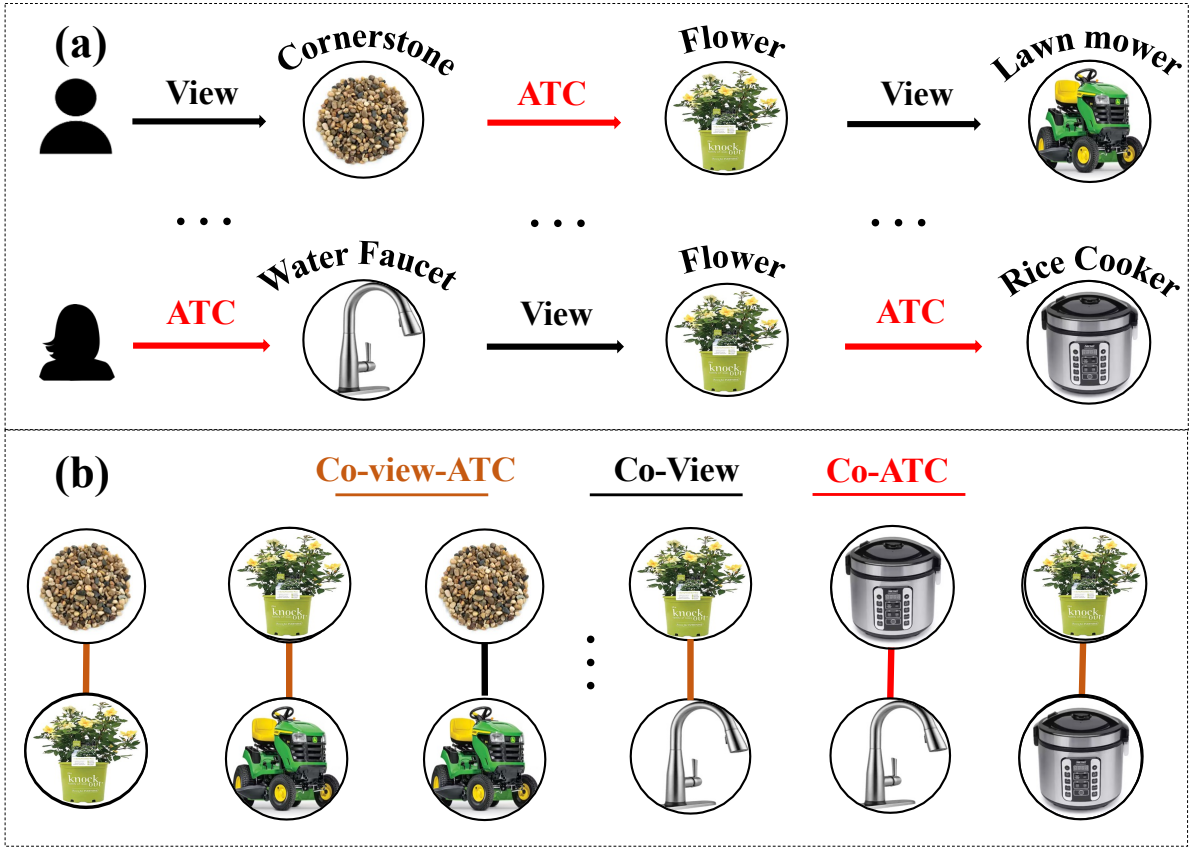




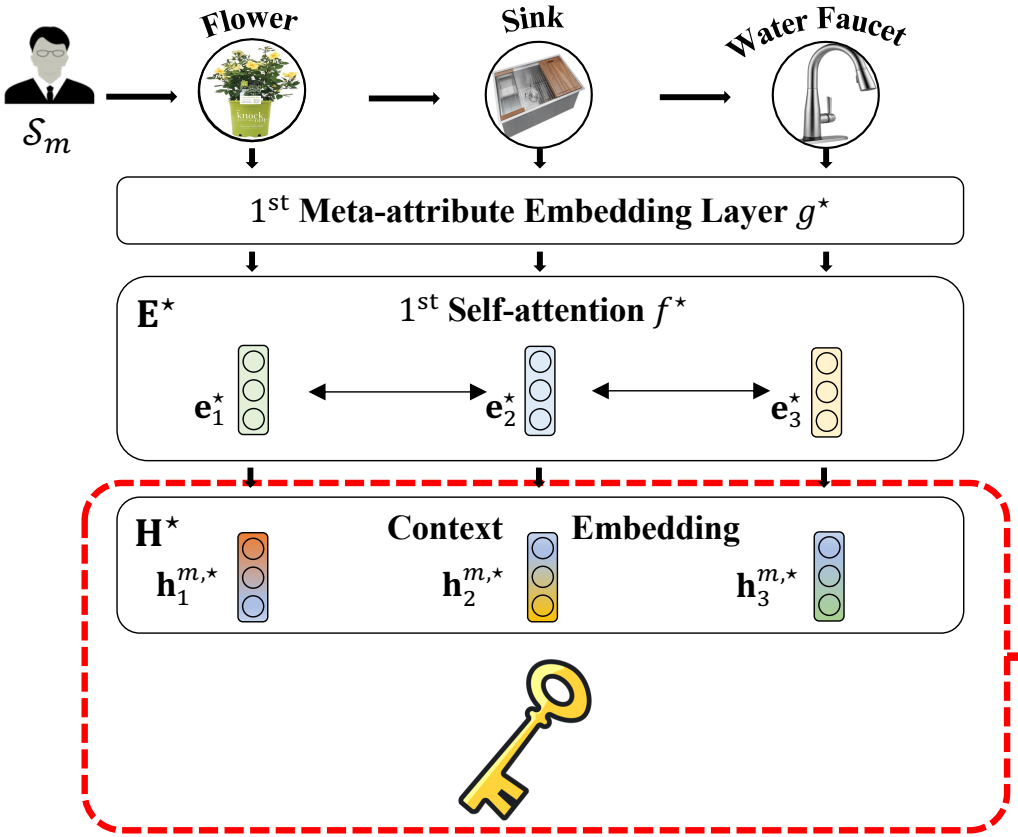


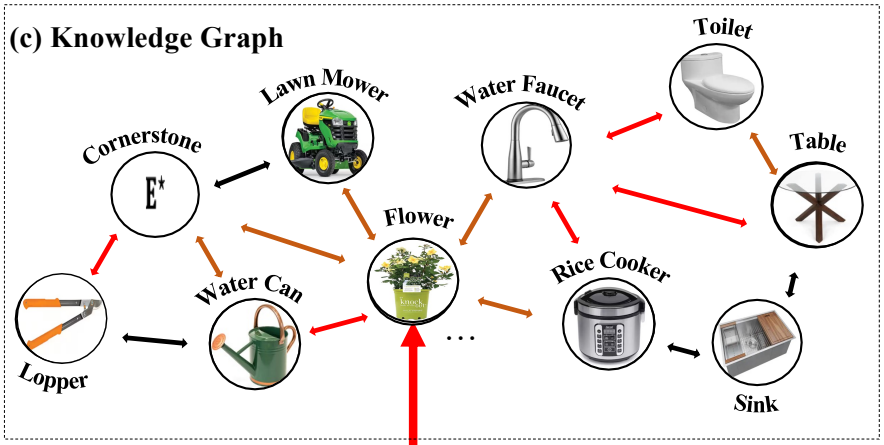
Link Prediction – Recommender Systems – Next Item

Items that **frequently co-appear** in the same session have **stronger relations**



Link Prediction – Recommender Systems – Next Item

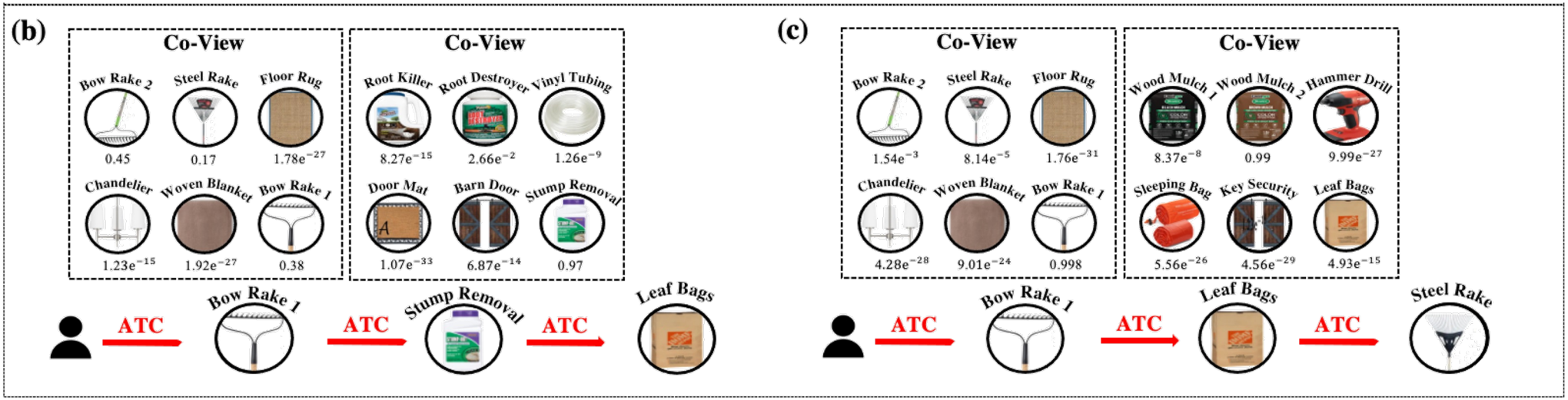




Query and Retrieval

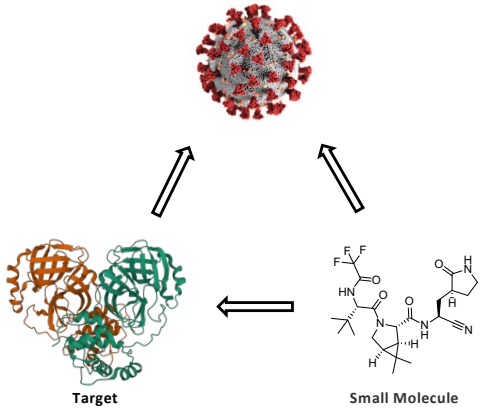
Link Prediction – Recommender Systems – Next Item

Effectiveness of Session-adaptive Message-passing



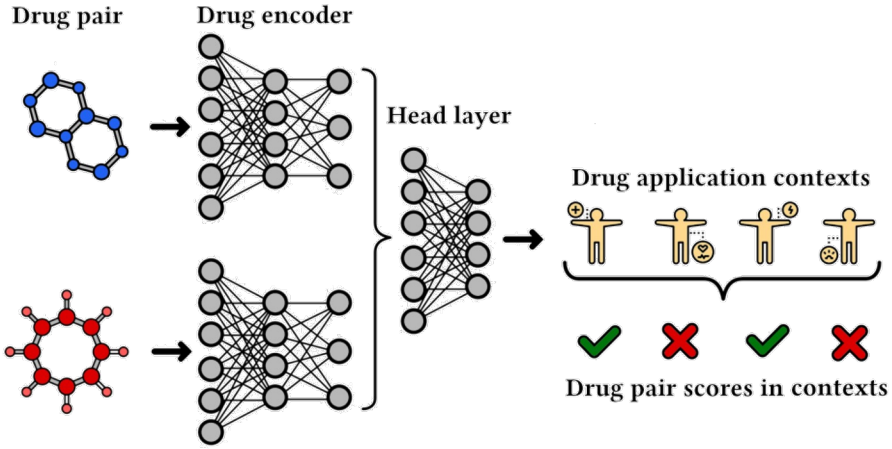
Graph Classification and Regression - Motivation

Drug Discovery



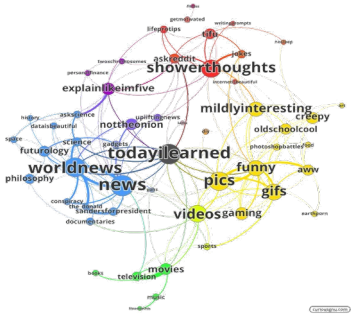
[1] Liana Zucco et al. (2020)

Drug-Drug Interaction



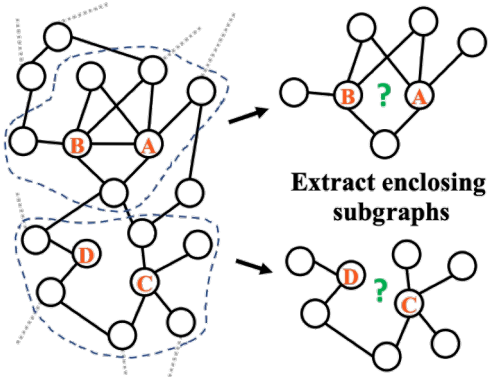
[2] Rozemberczik et al. (2022)

Social Topic Classification



[3] Hamilton et al. (2017)

Link Prediction

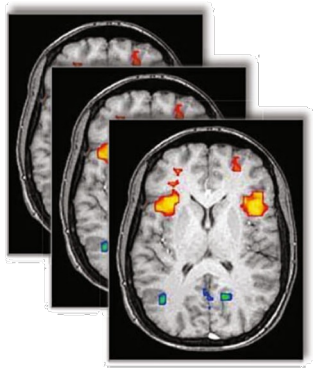


[4] Zhang et al. (2018)

Image Classification

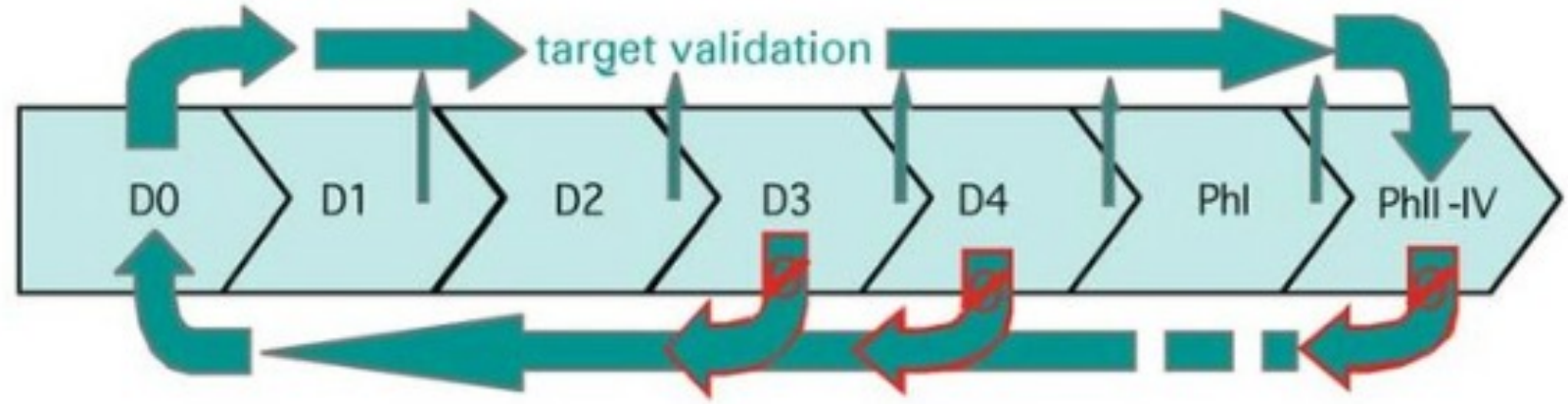
[5] Vasudevan et al. (2022)

Brain Classification



[6] Bi et al. (2020)

Graph Classification and Regression - Motivation



- **Wet lab**

- HTS involves hands-on experiments in a wet lab, where biological samples, drugs, and fluids are tested using chemicals and liquids.

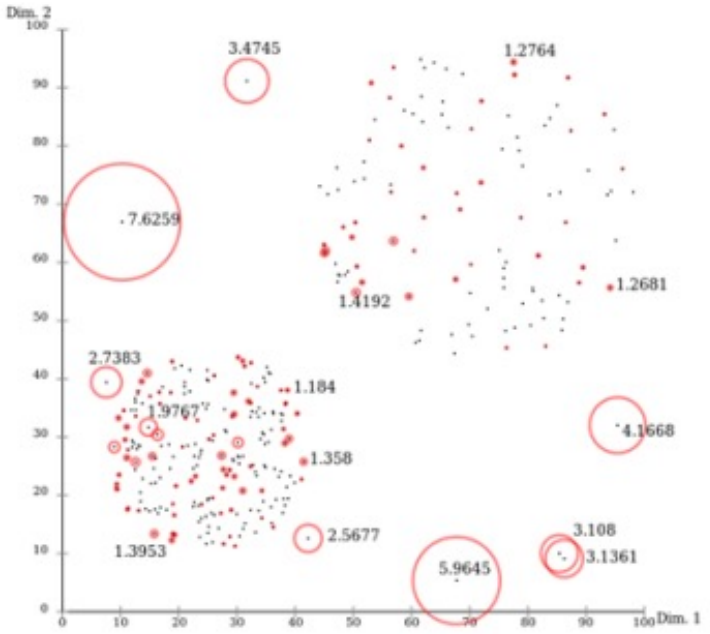
- **Dry lab**

- HTS also involves data analysis in a dry lab, where computer systems, coding, and data are used

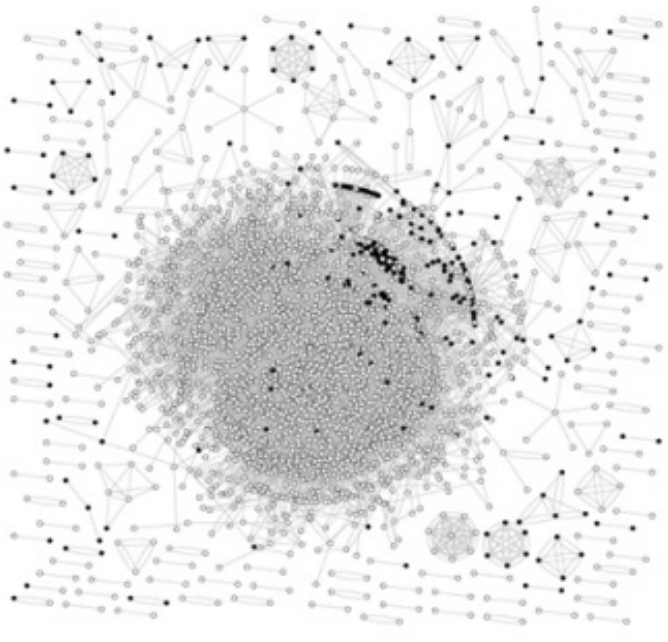
D0: Basic sciences, target selection.
D1: Assay development for high-throughput screening in vitro.
D2: High-throughput screening of public and proprietary compound libraries, ligand finding (hits).
D3: Lead optimization by medicinal chemistry, in vitro and in vivo models, initial pharmacokinetics and safety.
D4: Preparation for human studies: biomarkers, extensive pharmacokinetics, safety, metabolism in animals, formulation, chemical up-scaling.

Graph Classification and Regression - Motivation

Anomaly Detection

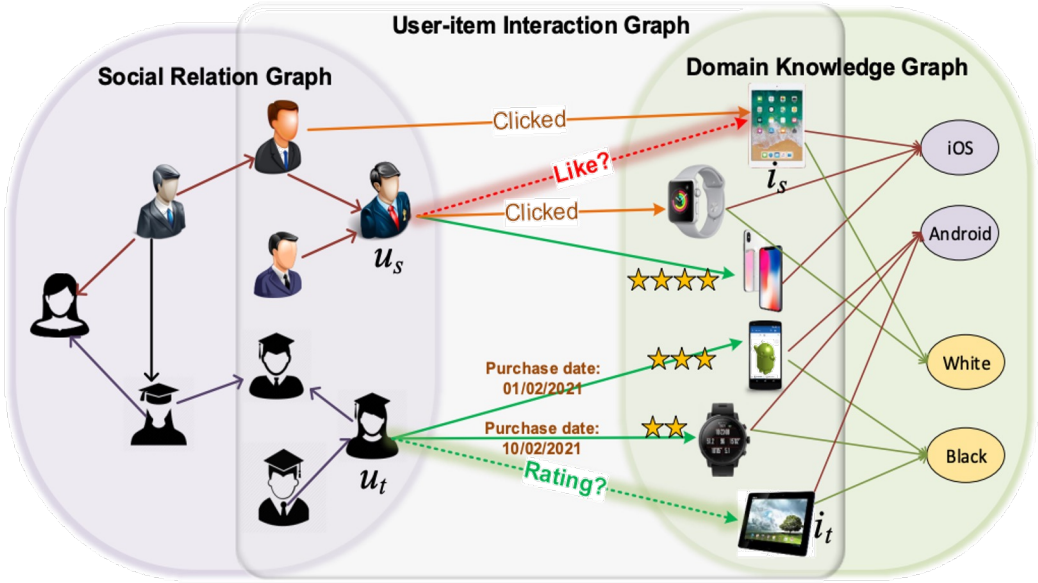


(a) Clouds of points (multi-dimensional)

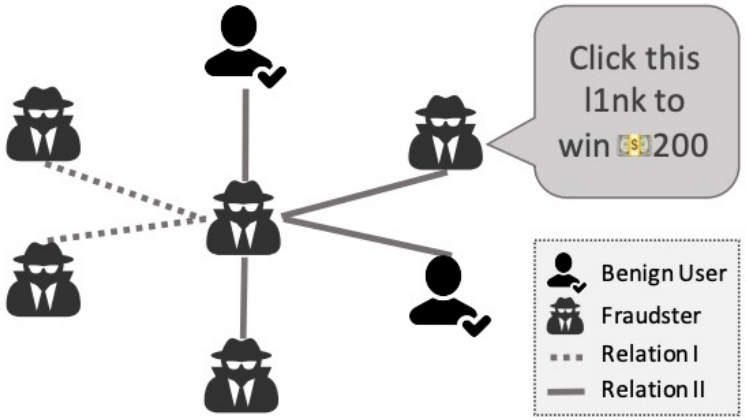


(b) Inter-linked objects (network)

Graph Classification and Regression



Review



Camouflage

Graph Classification and Regression

Data | **Model** | **Loss** | **Optimization**

```
# Load the dataset
dataset = TUDataset(root='data/TUDataset', name='MUTAG')  # MUTAG is an example; you can choose another TU dataset
```

```
dataset[0]
```

```
Data(edge_index=[2, 38], x=[17, 7], edge_attr=[38, 4], y=[1])
```

The MUTAG dataset consists of 188 chemical compounds divided into two classes according to their mutagenic effect on a bacterium.

Node labels:

0 C
1 N
2 O
3 F
4 I
5 Cl
6 Br

Edge labels:

0 aromatic
1 single
2 double
3 triple

Graph Classification and Regression

Data | **Model** | **Loss** | **Optimization**

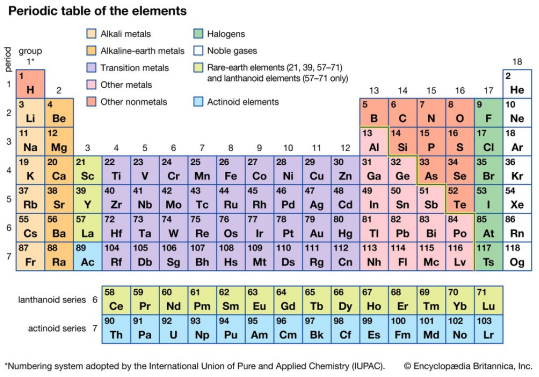
The MUTAG dataset consists of 188 chemical compounds divided into two classes according to their mutagenic effect on a bacterium.

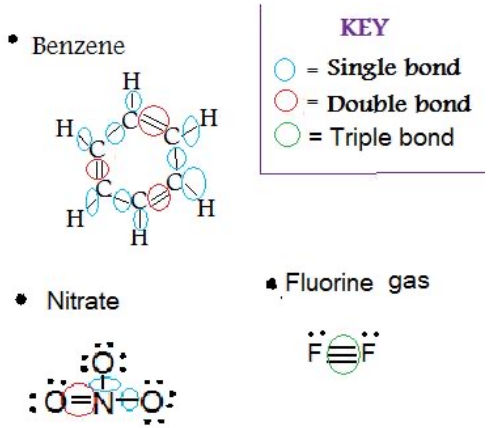
Node labels:

0 C
1 N
2 O
3 F
4 I
5 Cl
6 Br

Edge labels:

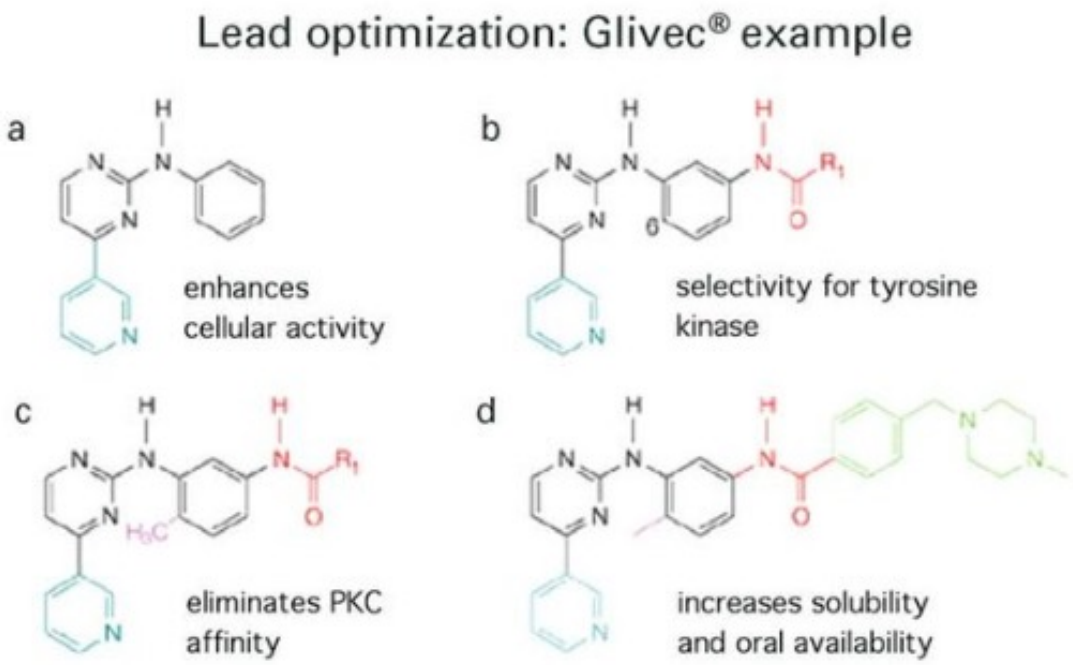
0 aromatic
1 single
2 double
3 triple





Graph Classification and Regression

Data	Model	Loss	Optimization



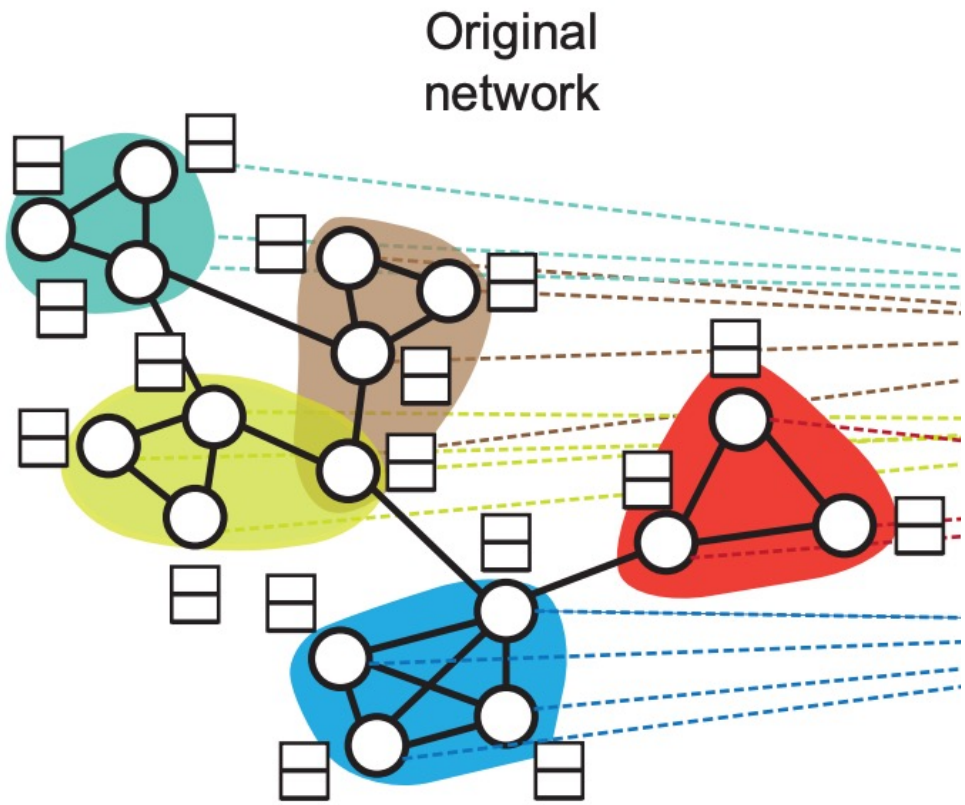
Cellular activity refers to the chemical activity of a cell, and includes how cells respond to and adapt to their environment

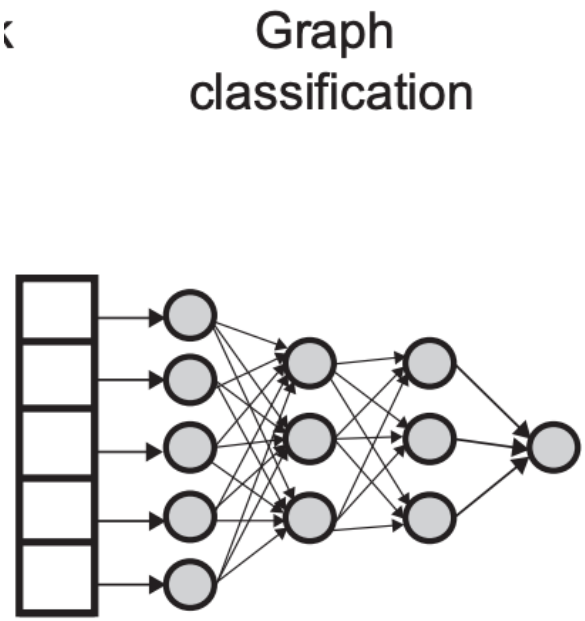
Tyrosine kinases are enzymes that play a key role in many cellular functions, including cell division, metabolism, and cell-to-cell communication

directed therapy for chronic amyotrophic leukemia (CML).

Sub-structure is really really very important

Graph Classification and Regression





$$h_v^{(k)} = \mathrm{ReLU}\left(W \cdot \mathrm{MEAN}\left\{h_u^{(k-1)},\ \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right). \quad \textbf{Local Structure}$$

$$h_G = \mathrm{READOUT}\left(\left\{h_v^{(K)} \mid v \in G\right\}\right). \quad \textbf{Global Structure}$$

Code Demo

Graph Classification and Regression

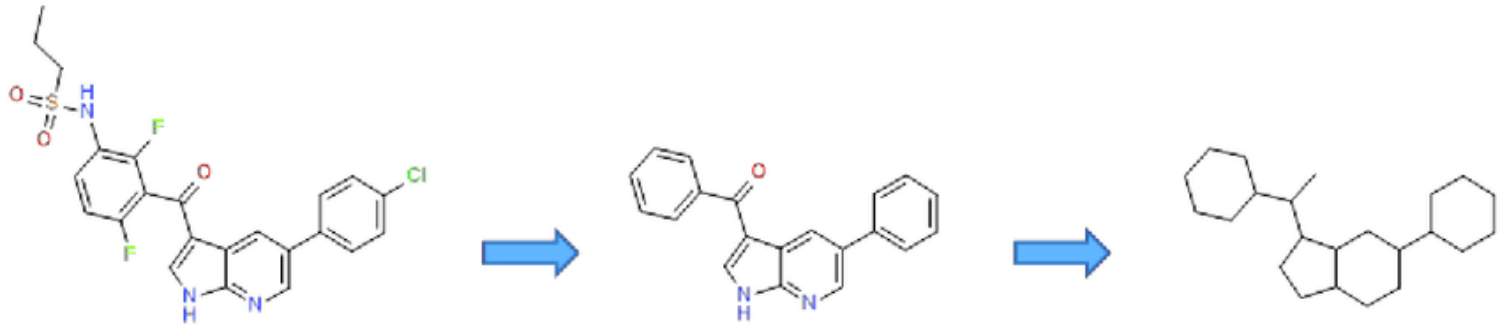
	Datasets	IMDB-B	IMDB-M	RDT-B	RDT-M5K	COLLAB	MUTAG	PROTEINS	PTC	NCI1
Datasets	# graphs	1000	1500	2000	5000	5000	188	1113	344	4110
	# classes	2	3	2	5	3	2	2	2	2
	Avg # nodes	19.8	13.0	429.6	508.5	74.5	17.9	39.1	25.5	29.8
Baselines	WL subtree	73.8 ± 3.9	50.9 ± 3.8	81.0 ± 3.1	52.5 ± 2.1	78.9 ± 1.9	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	$\mathbf{86.0 \pm 1.8}$ *
	DCNN	49.1	33.5	–	–	52.1	67.0	61.3	56.6	62.6
	PATCHYSAN	71.0 ± 2.2	45.2 ± 2.8	86.3 ± 1.6	49.1 ± 0.7	72.6 ± 2.2	$\mathbf{92.6 \pm 4.2}$ *	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
	DGCNN	70.0	47.8	–	–	73.7	85.8	75.5	58.6	74.4
	AWL	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 2.5	54.7 ± 2.9	73.9 ± 1.9	87.9 ± 9.8	–	–	–
GNN variants	SUM–MLP (**GIN-0**)	$\mathbf{75.1 \pm 5.1}$	$\mathbf{52.3 \pm 2.8}$	$\mathbf{92.4 \pm 2.5}$	$\mathbf{57.5 \pm 1.5}$	$\mathbf{80.2 \pm 1.9}$	$\mathbf{89.4 \pm 5.6}$	$\mathbf{76.2 \pm 2.8}$	$\mathbf{64.6 \pm 7.0}$	$\mathbf{82.7 \pm 1.7}$
	SUM–MLP (**GIN-ϵ**)	$\mathbf{74.3 \pm 5.1}$	$\mathbf{52.1 \pm 3.6}$	$\mathbf{92.2 \pm 2.3}$	$\mathbf{57.0 \pm 1.7}$	$\mathbf{80.1 \pm 1.9}$	$\mathbf{89.0 \pm 6.0}$	$\mathbf{75.9 \pm 3.8}$	63.7 ± 8.2	$\mathbf{82.7 \pm 1.6}$
	SUM–1-LAYER	74.1 ± 5.0	$\mathbf{52.2 \pm 2.4}$	90.0 ± 2.7	55.1 ± 1.6	$\mathbf{80.6 \pm 1.9}$	$\mathbf{90.0 \pm 8.8}$	$\mathbf{76.2 \pm 2.6}$	63.1 ± 5.7	82.0 ± 1.5
	MEAN–MLP	73.7 ± 3.7	$\mathbf{52.3 \pm 3.1}$	50.0 ± 0.0	20.0 ± 0.0	79.2 ± 2.3	83.5 ± 6.3	75.5 ± 3.4	$\mathbf{66.6 \pm 6.9}$	80.9 ± 1.8
	MEAN–1-LAYER (GCN)	74.0 ± 3.4	51.9 ± 3.8	50.0 ± 0.0	20.0 ± 0.0	79.0 ± 1.8	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0
	MAX–MLP	73.2 ± 5.8	51.1 ± 3.6	–	–	–	84.0 ± 6.1	76.0 ± 3.2	64.6 ± 10.2	77.8 ± 1.3
	MAX–1-LAYER (GraphSAGE)	72.3 ± 5.3	50.9 ± 2.2	–	–	–	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5

Graph Classification in Drug Discovery

Tab. 1: Statistics of our 9 datasets in *WelQrate* dataset collection, which has coverage of various important drug targets, challenging but realistic low active percentages.

Target Class	BioAssay ID (AID)	Target	Compound Type	Number of Compounds	Number of Actives	Percent Active	Unique BM Scaffolds
G Protein-Coupled Receptor (GPCR)	435008*	Orexin 1 Receptor	Antagonist	307,660	176	0.057%	86,108
	1798	M1 Muscarinic Receptor	Allosteric Agonist	60,706	164	0.270%	30,079
	435034	M1 Muscarinic Receptor	Allosteric Antagonist	60,359	78	0.129%	39,909
Ion Channel	1843	Potassium Ion Channel Kir2.1	Inhibitor	288,277	155	0.054%	82,140C
	2258	KCNQ2 Potassium Channel	Potentiator	289,068	247	0.085%	82,247
	463087	Cav3 T-type Calcium Channel	Inhibitor	95,650	652	0.682%	40,066
Transporter	488997*	Choline Transporter	Inhibitor	288,564	236	0.082%	82,343
Kinase	2689*	Serine Threonine Kinase 33	Inhibitor	304,475	120	0.039%	85,314
Enzyme	485290	Tyrosyl-DNA Phosphodiesterase	Inhibitor	281,146	586	0.208%	80,984

* Indicates additional experimental measurements are available for those datasets. See Sec. 4.1 for details.



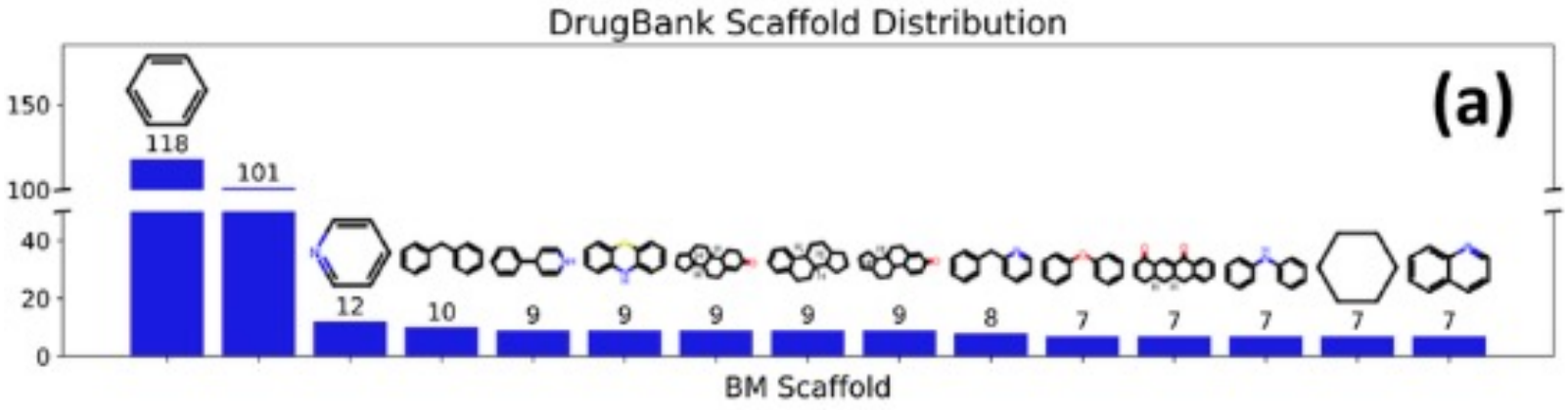
Graph Classification and Regression

Tab. 1: Statistics of our 9 datasets in *WelQrate* dataset collection, which has coverage of various important drug targets, challenging but realistic low active percentages.

Target Class	BioAssay ID (AID)	Target	Compound Type	Number of Compounds	Number of Actives	Percent Active	Unique BM Scaffolds
G Protein-Coupled Receptor (GPCR)	435008*	Orexin 1 Receptor	Antagonist	307,660	176	0.057%	86,108
	1798	M1 Muscarinic Receptor	Allosteric Agonist	60,706	164	0.270%	30,079
	435034	M1 Muscarinic Receptor	Allosteric Antagonist	60,359	78	0.129%	39,909
Ion Channel	1843	Potassium Ion Channel Kir2.1	Inhibitor	288,277	155	0.054%	82,140C
	2258	KCNQ2 Potassium Channel	Potentiator	289,068	247	0.085%	82,247
	463087	Cav3 T-type Calcium Channel	Inhibitor	95,650	652	0.682%	40,066
Transporter	488997*	Choline Transporter	Inhibitor	288,564	236	0.082%	82,343
Kinase	2689*	Serine Threonine Kinase 33	Inhibitor	304,475	120	0.039%	85,314
Enzyme	485290	Tyrosyl-DNA Phosphodiesterase	Inhibitor	281,146	586	0.208%	80,984

* Indicates additional experimental measurements are available for those datasets. See Sec. 4.1 for details.

It ensures **realistic model evaluation**, promotes **chemical diversity** and encourages the discovery of **novel scaffolds and first-in-class drugs**



Graph Classification and Regression

Tab. 1: Statistics of our 9 datasets in *WelQrate* dataset collection, which has coverage of various important drug targets, challenging but realistic low active percentages.

Target Class	BioAssay ID (AID)	Target	Compound Type	Number of Compounds	Number of Actives	Percent Active	Unique BM Scaffolds
G Protein-Coupled Receptor (GPCR)	435008*	Orexin 1 Receptor	Antagonist	307,660	176	0.057%	86,108
	1798	M1 Muscarinic Receptor	Allosteric Agonist	60,706	164	0.270%	30,079
	435034	M1 Muscarinic Receptor	Allosteric Antagonist	60,359	78	0.129%	39,909
Ion Channel	1843	Potassium Ion Channel Kir2.1	Inhibitor	288,277	155	0.054%	82,140C
	2258	KCNQ2 Potassium Channel	Potentiator	289,068	247	0.085%	82,247
	463087	Cav3 T-type Calcium Channel	Inhibitor	95,650	652	0.682%	40,066
Transporter	488997*	Choline Transporter	Inhibitor	288,564	236	0.082%	82,343
Kinase	2689*	Serine Threonine Kinase 33	Inhibitor	304,475	120	0.039%	85,314
Enzyme	485290	Tyrosyl-DNA Phosphodiesterase	Inhibitor	281,146	586	0.208%	80,984

* Indicates additional experimental measurements are available for those datasets. See Sec. 4.1 for details.

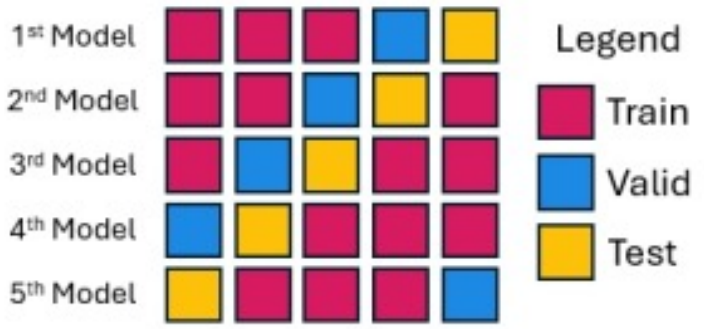


Fig. 3: Illustration of the adapted cross-valiation.

Graph Classification and Regression

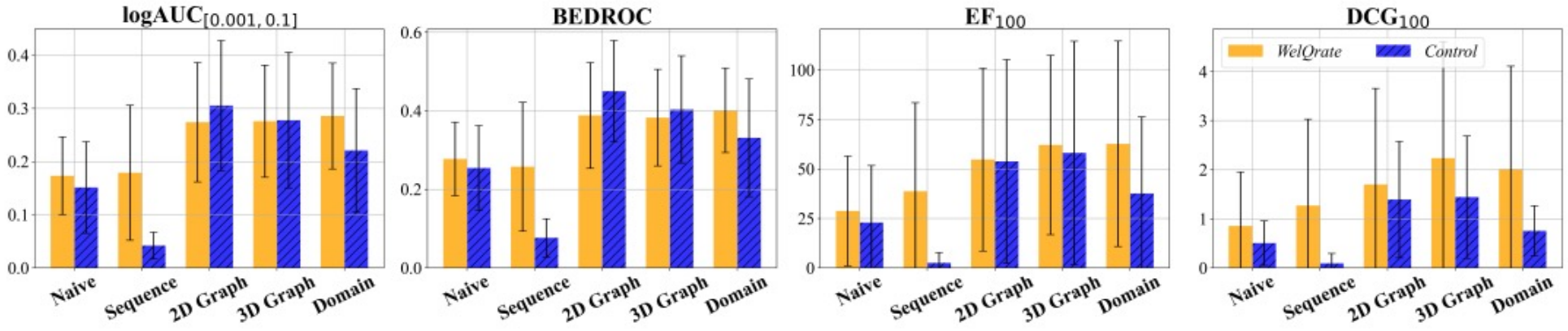
Node atom number, chemical bonds

- **Sequence-Based:** SMILES2Vec [31], TextCNN [32].
- **2D Graph-Based:** Graph Convolutional Neural Network (GCN) [33], Graph Isomorphism Network (GIN) [34], Graph Attention Network (GAT) [35].
- **3D Graph-Based:** SchNet [36], DimeNet++ [37], SphereNet [38].

Node atom number, Node 3D coordinates

- **Naive Baseline:** Atomic-level Pooling averages the atomic features as molecular representations.
- **Domain Baseline:** Molecular-level Descriptor, specifically, the BioChemical Library (BCL) [39] is utilized to extract a domain-driven descriptor set, such as signed 2D and 3D autorcorrelations [23] (with details in the supplement Sec. C.1).

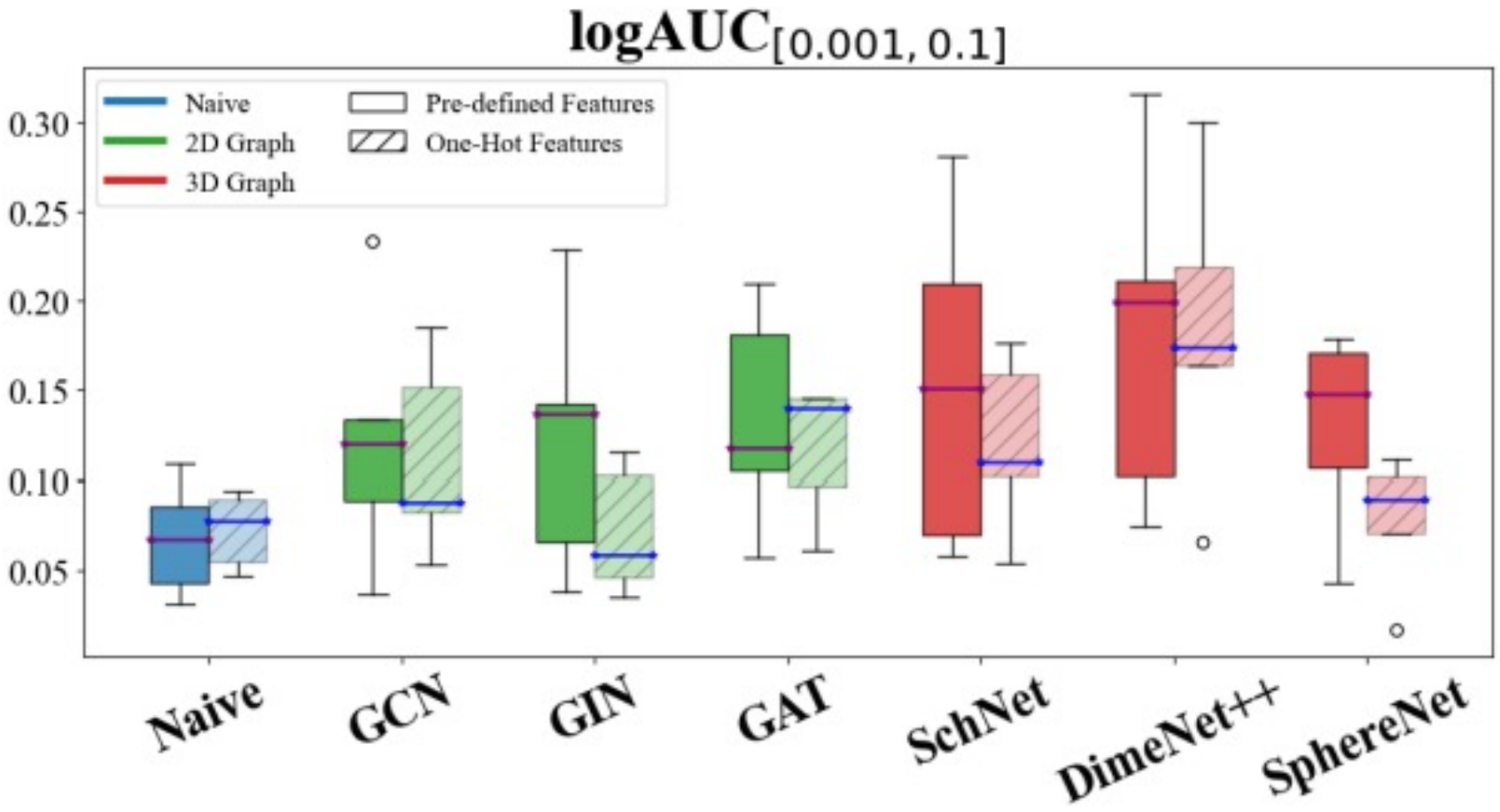
Graph Classification and Regression



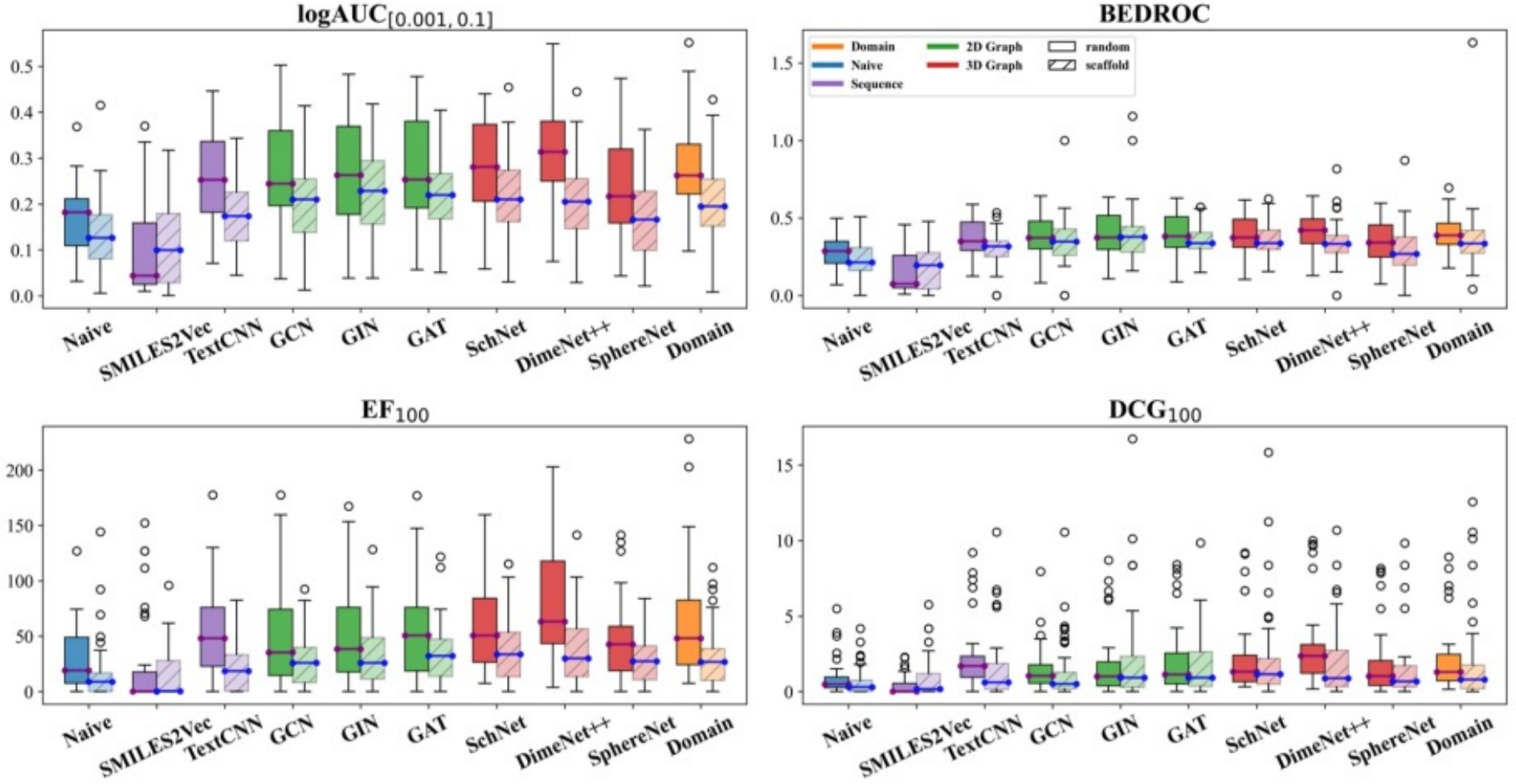
Domain Performs pretty good

3D Graph in general is better than 2D Graph than others

Graph Classification and Regression



Graph Classification and Regression



Question Time!