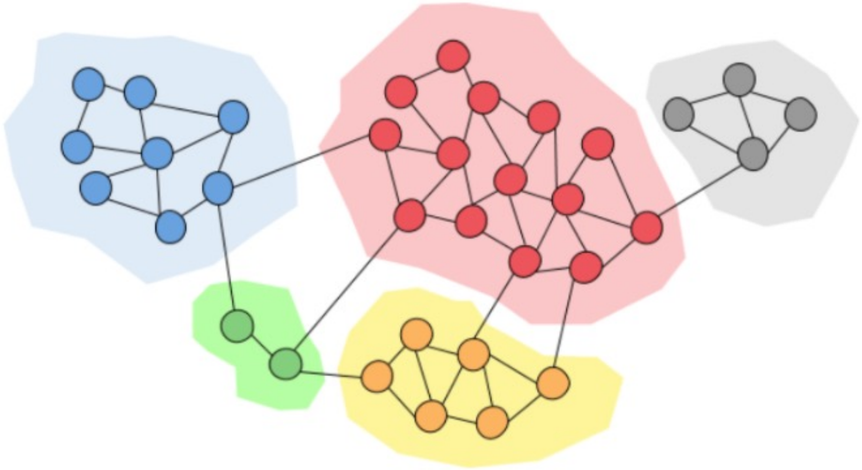
A Deep Learning Framework for Self-evolving Hierarchical Community Detection

Daizong Ding[1] Mi Zhang[1] Hanrui Wang[1] Xudong Pan[1] Min Yang[1] Xiangnan He[2]

1. School of Computer Science, Fudan University
2. School of Data Science, University of Science and Technology of China

Community Detection



Input

- A graph represented by nodes and edges

Output

- Node-community affiliations

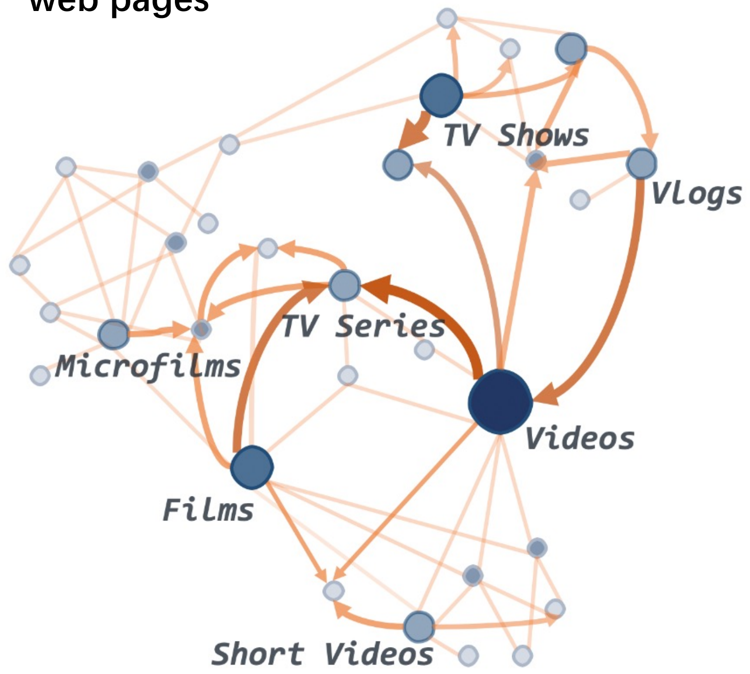
Application

- Social network analysis
- Information retrieval
- ……

Image from: https://towardsdatascience.com/community-detection-algorithms-9bd8951e7dae

Hierarchical Community Detection

web pages



Motivation

- Complex networks often have hierarchical structures

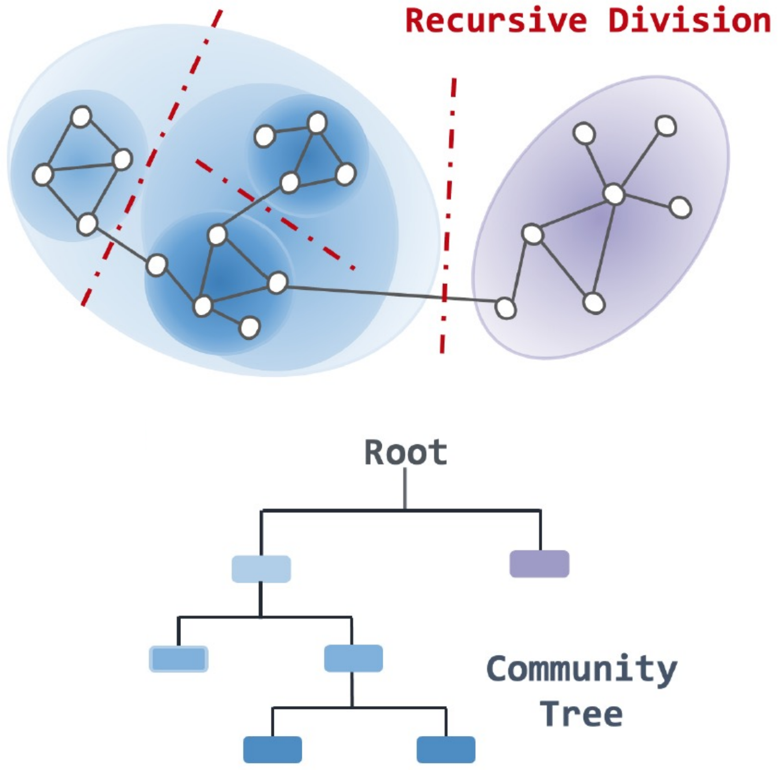
Design

- Modeling hierarchical relationships between clusters
- Community tree

Methods

- Louvain
- Label Propagation Algorithm
- ……

Heuristic Algorithms



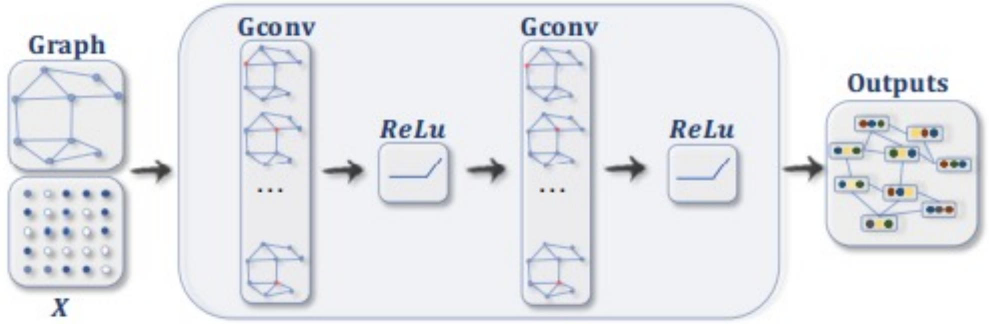
Solution

- Recursively division
  - Louvain
- Recursively aggregation

Cornerstone

- Heuristic algorithms
  - Random search
  - Greedy strategy

Deep Neural Networks



Recently,

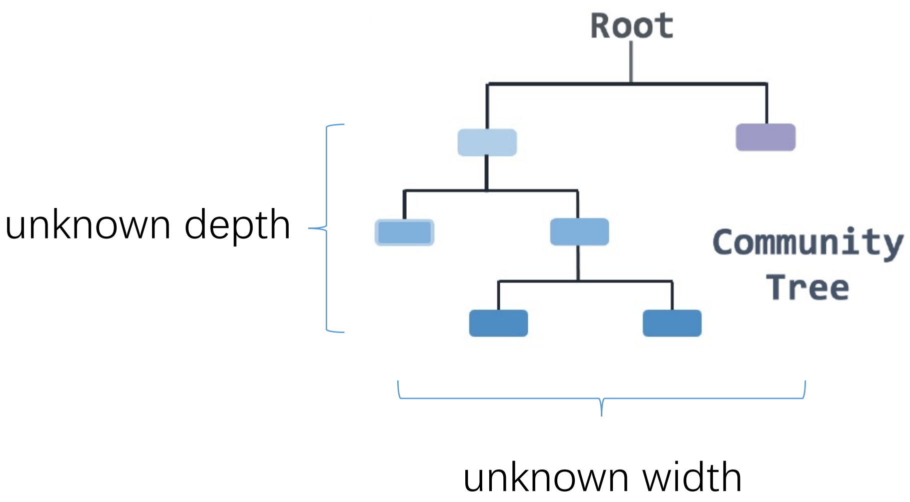
- Deep neural network (DNN) has been applied to various graph applications
  - Node classification
  - Link prediction

However,

- Such technique has not been validated on hierarchical community detection
- Why?

Image from: Wu Z, Pan S, Chen F, et al. A comprehensive survey on graph neural networks[J]. IEEE transactions on neural networks and learning systems, 2020, 32(1): 4-24.

Problem Analysis



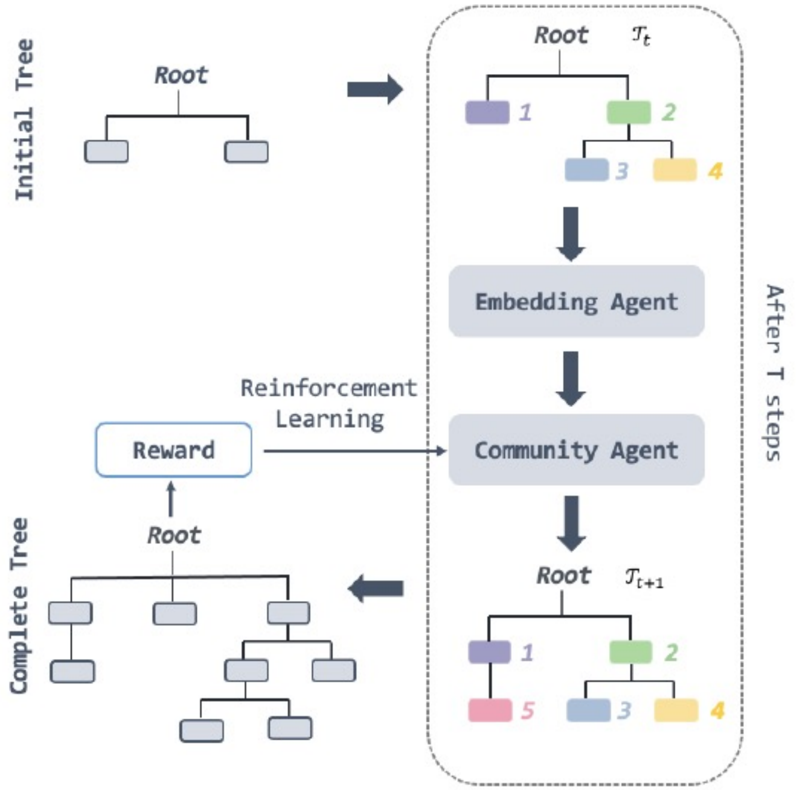
Main reason:

- DNN requires parametrized inputs and outputs:
    - Label of a node (an integer ID)
    - Link between two nodes (0 or 1)

However,

- It is difficult to parametrize a community tree without knowing its width and depth

Overview



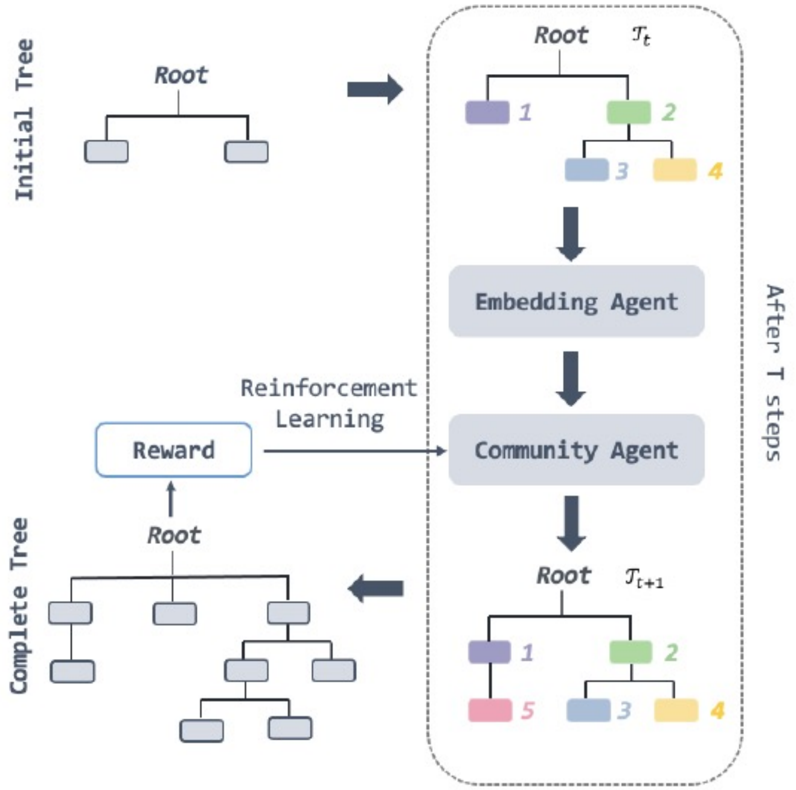
Our solution: ReinCom

- Dividing the problem into sub-problems that can be parametrized by the DNN

Tree generation by DNN:

- At each step, the DNN outputs the position for inserting a new community
- Starting from a small community tree, we can obtain a large one after several iteration
- Leverage reinforcement learning to guide the generation

Overview



Embedding Agent

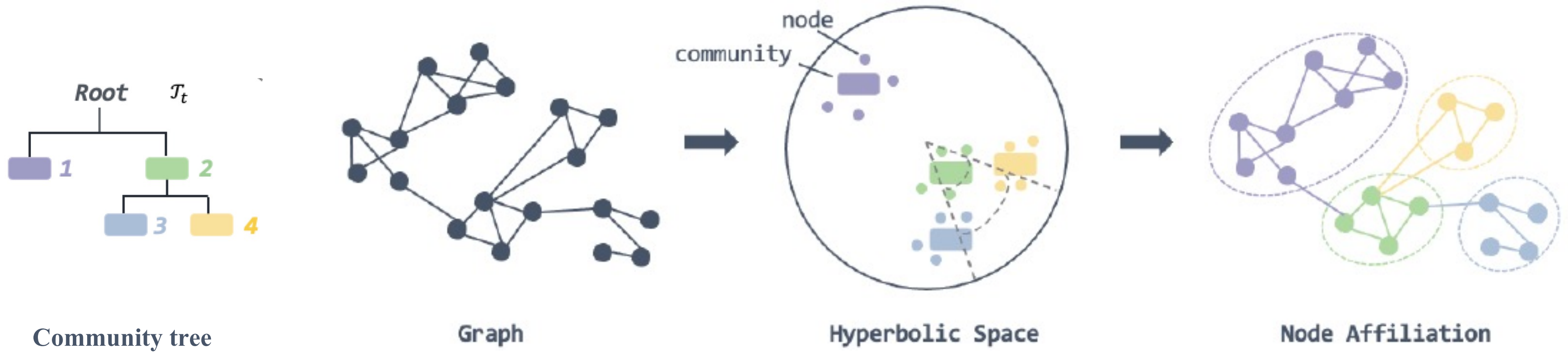
- Given a community tree, it partitions nodes to different communities
- Measure the quality of current community tree

Community Agent

- Adjust the existing community tree
- Predict the next position for inserting a new community
- Build new community tree and pass it to the embedding agent

Framework

- Two agents are designed to work collaboratively.



Input:

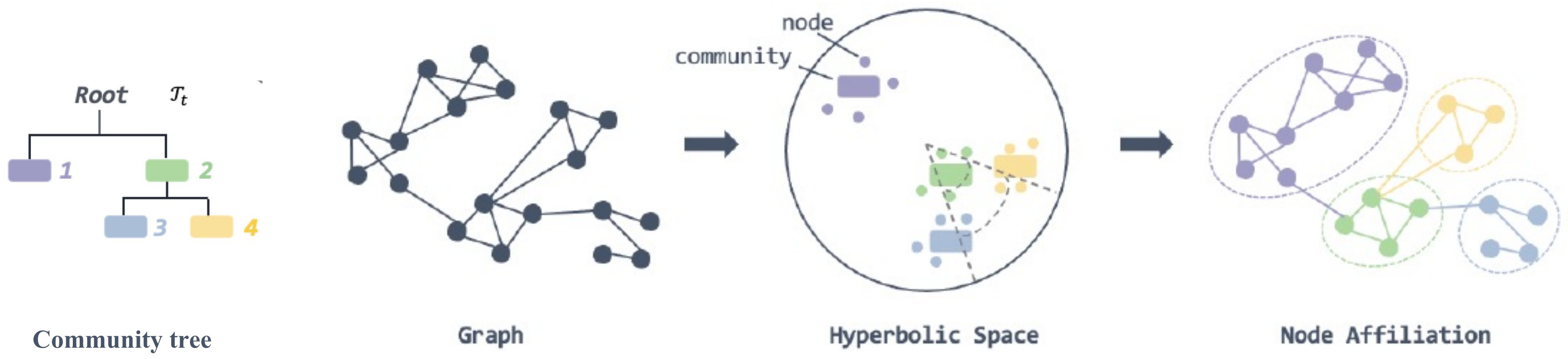
- The community tree $\mathcal{T}_t$ at time t, containing communities $c \in \{1, \cdots, t\}$
- The graph $\mathcal{G} = (\mathcal{V}, \Upsilon)$, where $\mathcal{V}$ is the set of nodes and Υ represents linkage information between nodes

Output:

- For each node v_i, the agent outputs its community c_i

Goal:

- Estimate the quality of the community tree $\mathcal{T}_t$



Embedding space:

- For each node v_i, we embed it with a vector representation $e_i \in \mathbb{R}^D$
- For each cluster c, we also map it to the same vector space and assign it with $e_c \in \mathbb{R}^D$

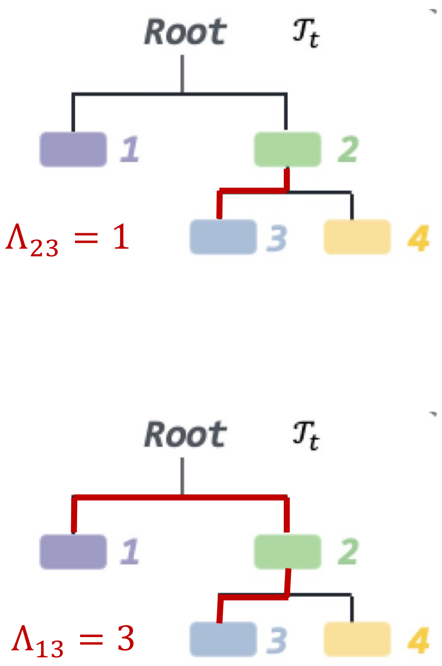
Node-community affiliation:

- The probability of node v_i belongs to community c can be measured by,

$$\rho_{ic} \propto \exp(-\|e_i - e_c\|^2)$$

- How to learn ρ_{ic}?

Community tree



Distance on the community tree:

- We first define the distance between two communities given $\mathcal{T}_t$
- The length of path to the common ancestor

Distance between two nodes:

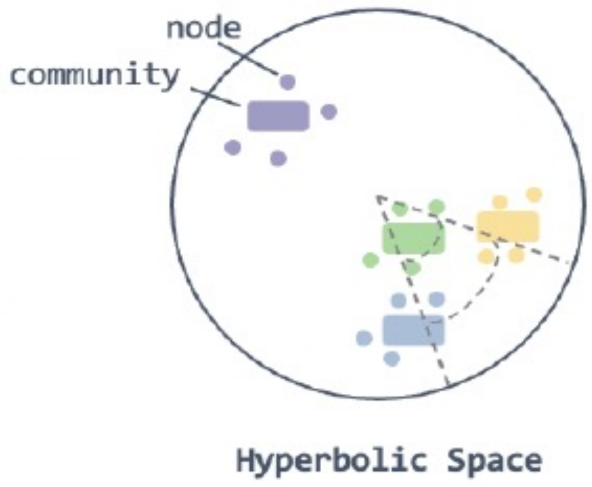
- Then the distance between two nodes can be represented by,

$$d_{ij} = \mathbb{E}_{c_i \sim p(c|v_i), c_j \sim p(c|v_j)}\left[\Lambda\left(c_i, c_j\right)\right] = \rho_i^T \Lambda \rho_j$$

Learning goal:

$$\sum_{v_i, v_j, v_k} \max\left(0, \beta + d_{ik} - d_{ij}\right)$$

- where v_i and v_j have edge, while v_i and v_k do not have edge



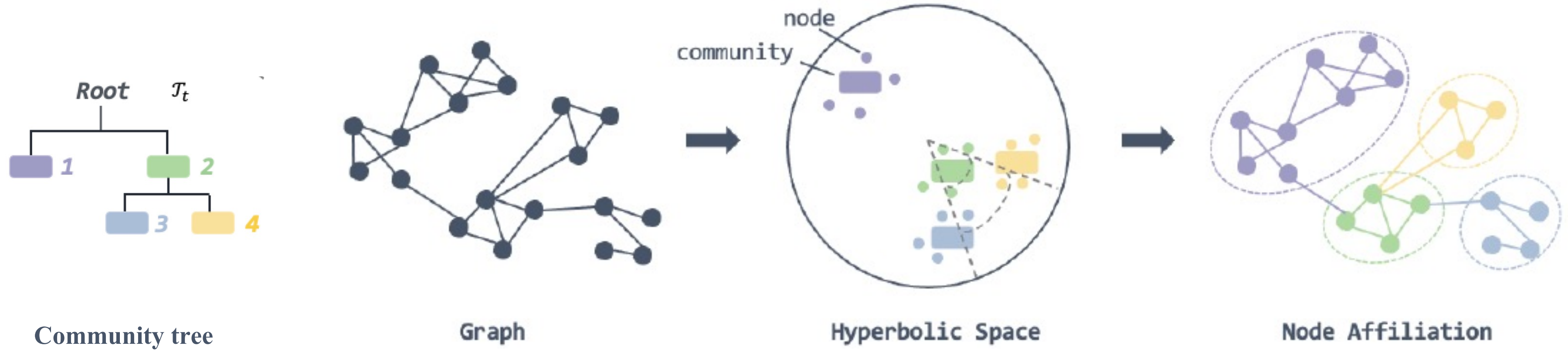
The hyperbolic embedding space:

- To better model the hierarchical structure, we leverage the hyperbolic space for the embeddings:
  - $\|e\|_2 \leq 1$
  - $\|e_i - e_c\|_H^2 = \operatorname{arccosh}\left(1 + \frac{2\|e_i - e_c\|_2^2}{(1-\|e_i\|_2)(1-\|e_c\|_2)}\right)$

To satisfy the constraint:

$$e_i = \left(1 - \exp\left(-\omega(\eta_i)\right)\right) \cdot \frac{\tilde{\boldsymbol{e}}_{\boldsymbol{i}}}{\|\tilde{\boldsymbol{e}}_{\boldsymbol{i}}\|_2}$$

- $\eta_i \in \mathbb{R}$ is the scale parameter
- $\tilde{\boldsymbol{e}}_{\boldsymbol{i}} \in \mathbb{R}^{\boldsymbol{D}}$ is the vector parameter

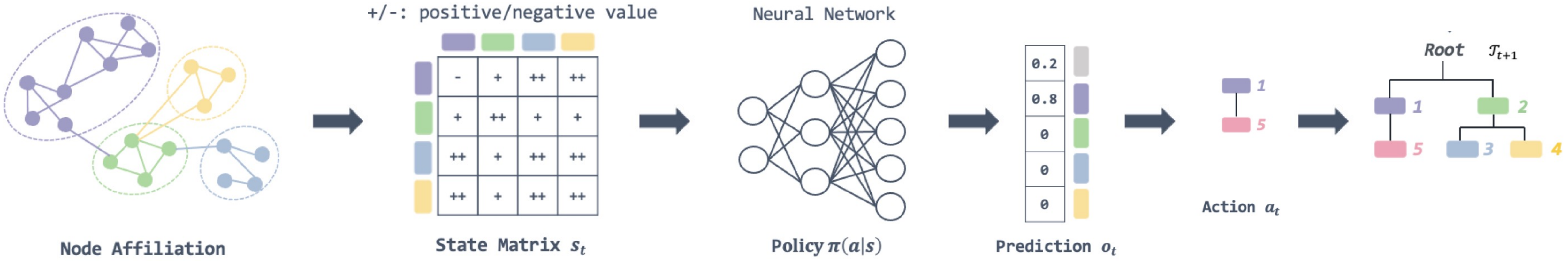


Overview:

- After the learning, the node-community affiliation c_i is calculated by,

$$c_i = argmax\ \rho_{ic}$$

- The hierarchical information in $\mathcal{T}_t$ is learned by the distance Λ on the tree, the graph $\mathcal{G}$ and the hyperbolic space

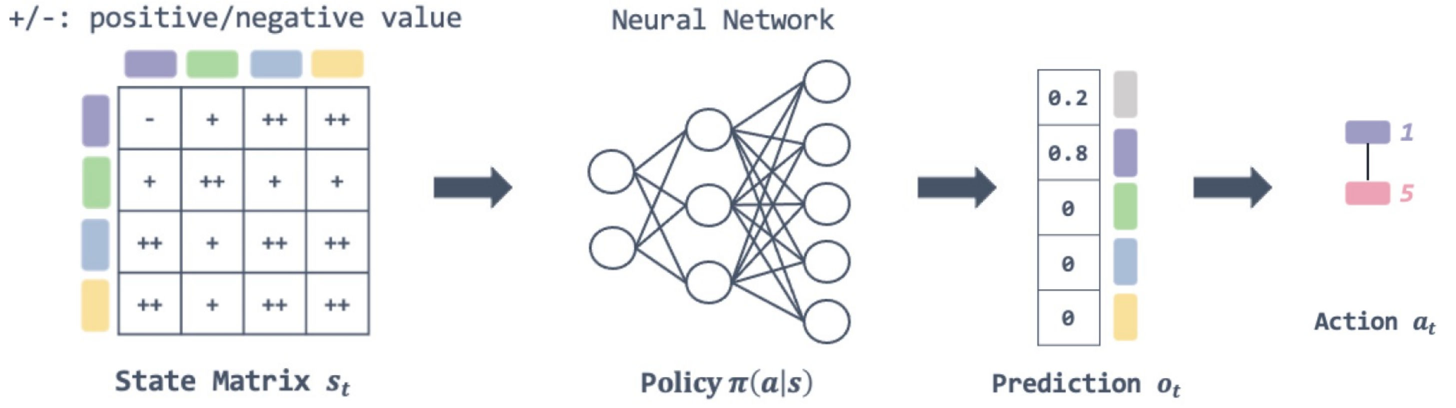


Overview:

- Predicting the position for inserting a new community according to the node-community affiliations
- Leveraging the DNN for the prediction

Why do we need to insert a new community?

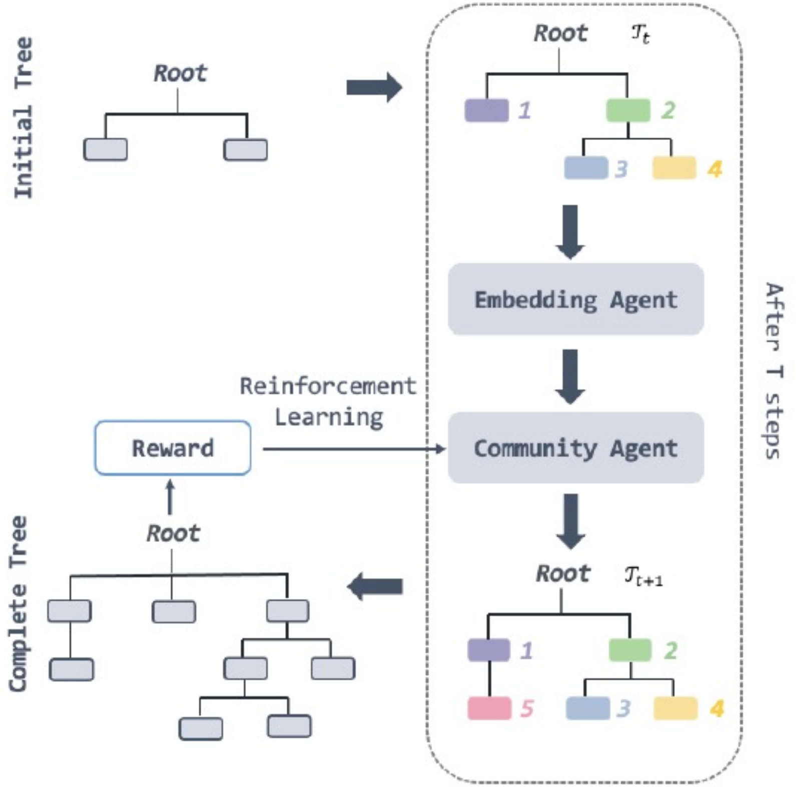
- Current community tree $\mathcal{T}_t$ is not effective enough
  - For node v_i and v_j with link between them, the c_i and c_j are different
  - For node v_i and v_j without a link, they have the same c
- We can describe the inaccurate node-community affiliations by a **state matrix**
- Nevertheless, it is difficult to determine the position when the state matrix becomes large and complex



Solution:

- We propose to leverage DNN for the prediction
- Given the state matrix s_t, the DNN predicts the probability o_t of existing communities
- Then we sample an $a_t = \{0,1,\cdots,t\}$ from o_t and insert a new community under a_t

Overview

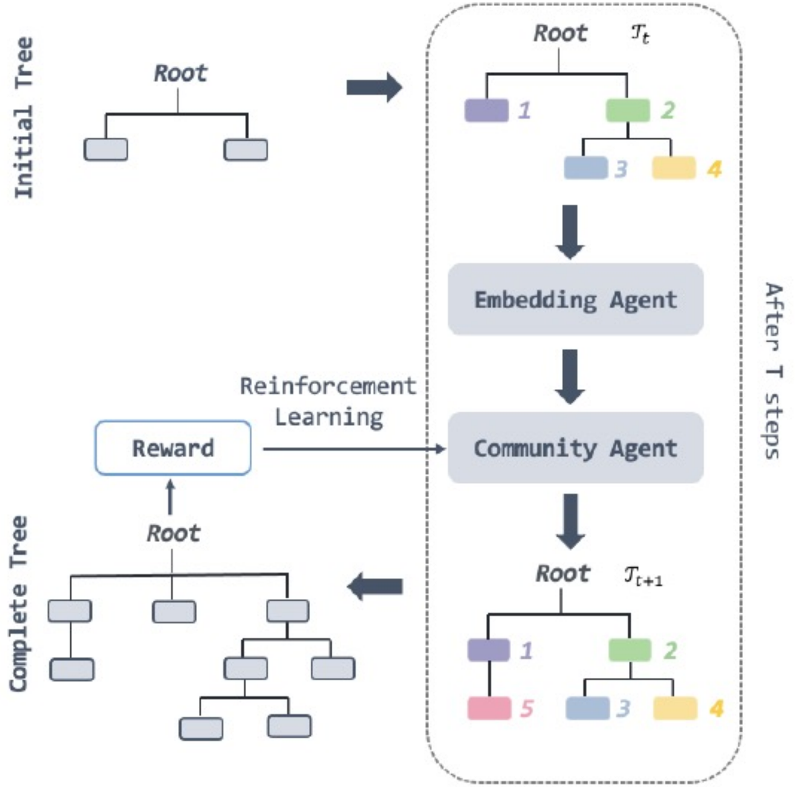


Generation of a community tree:

- Randomly initialize the embedding agent
- Set $\mathcal{T}_0 = \{0\}$
- For $t = 1, \cdots, T-1$:
  - Update the embedding agent with $\mathcal{T}_t$
  - Calculate the state matrix s_t with c_i
  - Use the community agent $\pi(a|s_t)$ to build $\mathcal{T}_{t+1}$
- Output the $\mathcal{T}_T$ and the latest c_i

How to train the community agent?

Reinforcement Learning



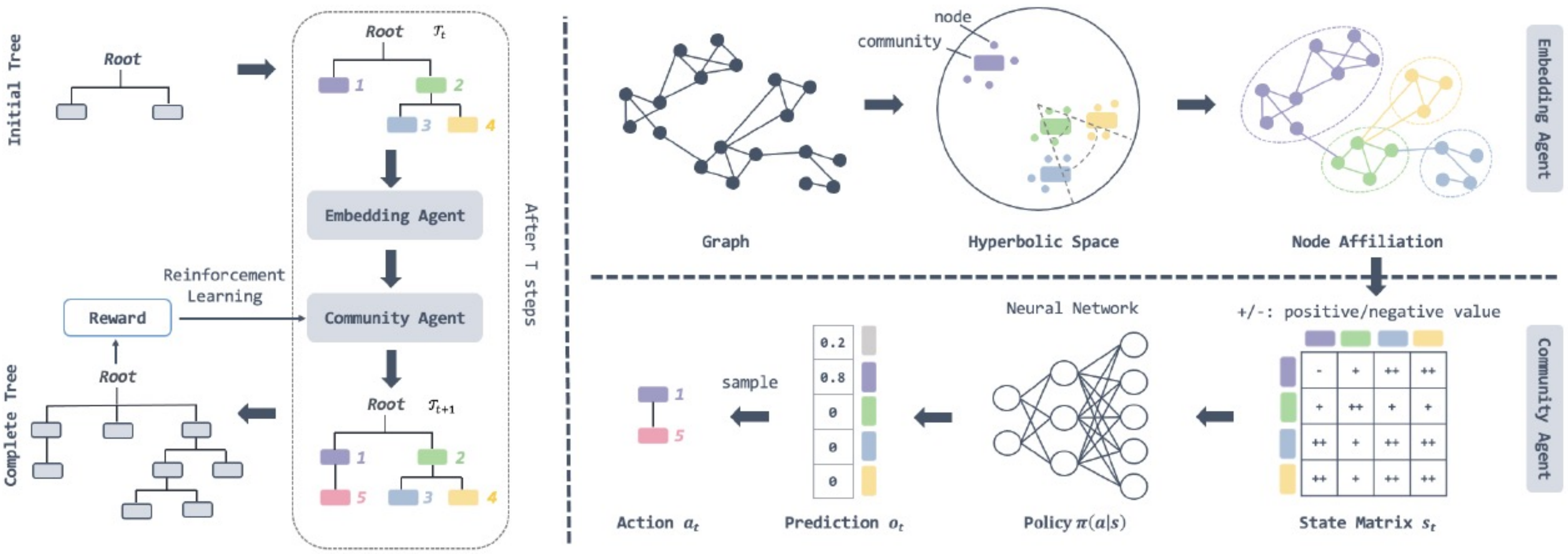
We leverage the reinforcement learning:

- Generating a community tree $\mathcal{T}_T$
- Calculate the reward of the tree $\mathcal{T}_t$ at each step by

$$r_t = \sum_{(v_i, v_k) \in y_{ik}=0} d_{ik} - \sum_{(v_i, v_j) \in y_{ij}=1} d_{ik}$$

- Update the community tree with $[r_1, \cdots, r_T]$

Overview


Initial Tree
Root
Root $\mathcal{T}_t$
Embedding Agent
Community Agent
Root $\mathcal{T}_{t+1}$
After T steps
Reinforcement Learning
Reward
Root
Complete Tree
node
community
Graph
Hyperbolic Space
Node Affiliation
Embedding Agent
+/-: positive/negative value
Neural Network
sample
0.2
0.8
0
0
0
Action a_t
Prediction o_t
Policy $\pi(a|s)$
State Matrix s_t
Community Agent

Experiment

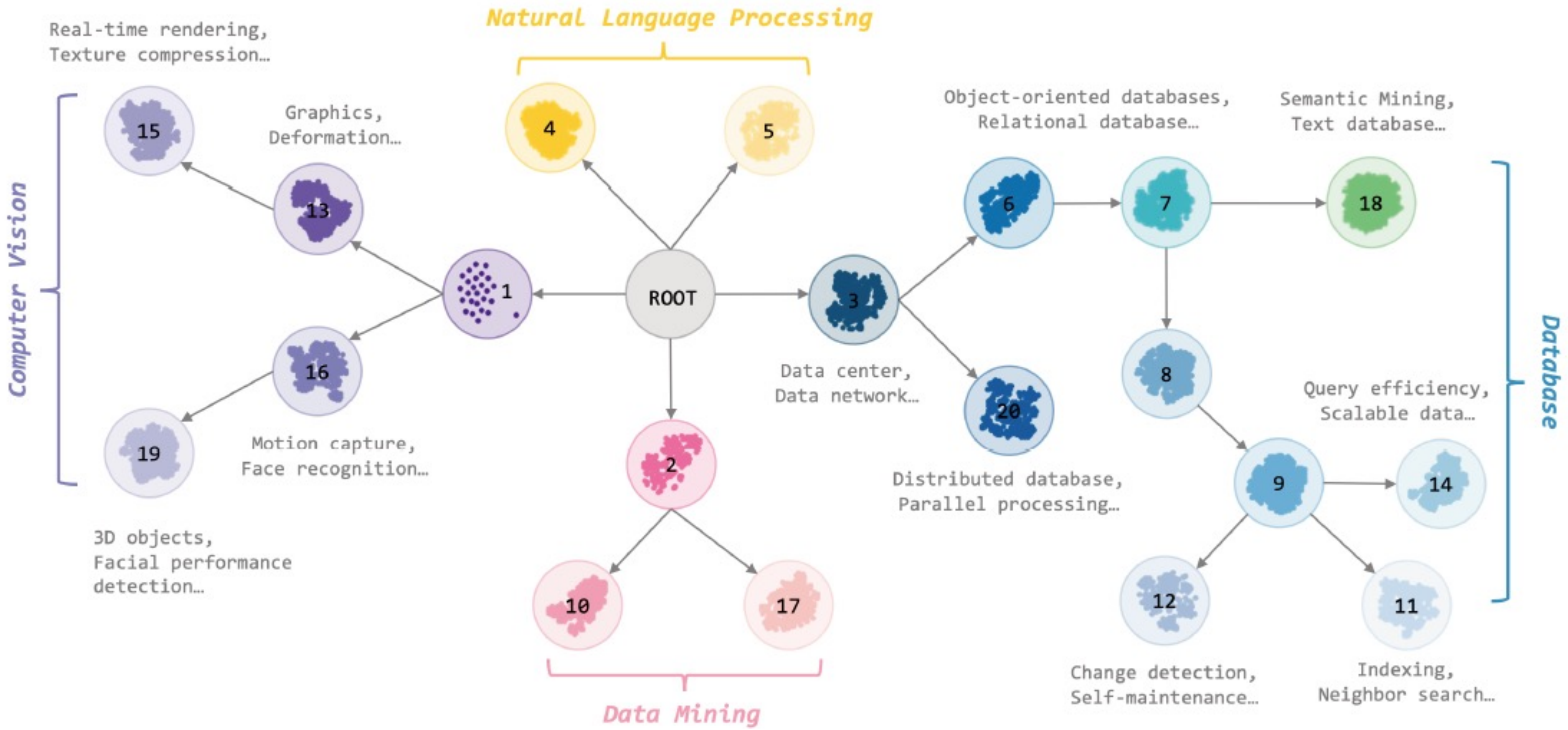
Table 1: Statistics of datasets.

	Aminer	BlogCatalog	Wiki-Vote	Deezer-RO
Nodes	12840	8943	3513	11847
Edges	190658	660840	95028	105844
Labels	4	39	NA	78
Modularity	✓	✓	✓	✓
NMI	✓	×	×	×
AUC	✓	✓	✓	✓
F1	×	✓	×	✓

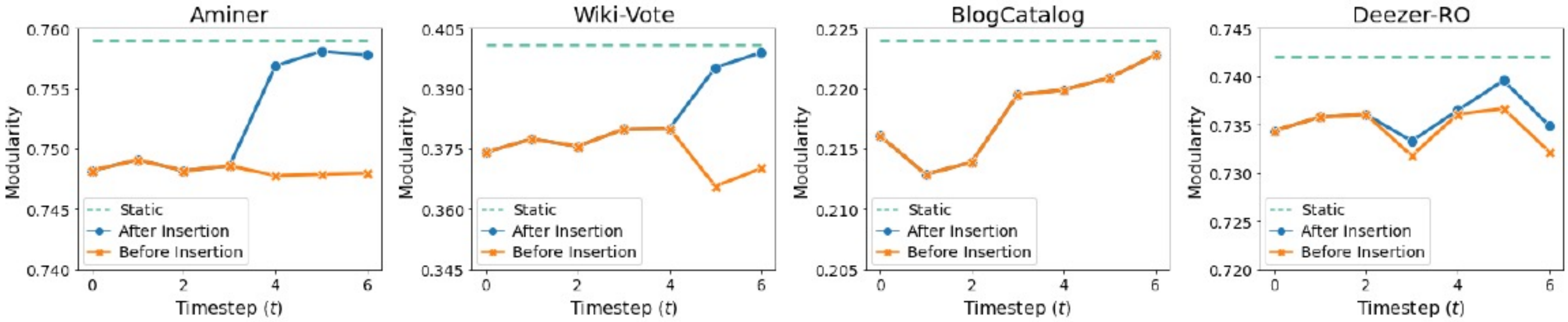
Main Results

		Modularity				NMI
		Aminer	Wiki-Vote	BlogCatalog	Deezer-RO	Aminer
	GEMSEC	0.661	0.211	0.021	0.649	0.361
hierarchical	Louvain	0.647	0.307	0.159	0.603	0.539
	HCDE	0.689	0.210	0.180	0.037	0.410
non-hierarchical	MNMF	0.709	0.297	0.154	0.665	0.294
	vGraph	0.710	0.258	N/A	N/A	0.001
	ComE	0.745	0.309	0.139	0.740	0.765
	ReinCom	**0.759**	**0.403**	**0.224**	**0.742**	**0.798**

Visualization



Online Updating



When there are new nodes and links:

- We can leverage the community agent to insert a new community

More Results

Table 3: F1 value for node classification.

	Deezer-RO		BlogCatalog	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1
LINE	0.023	0.302	0.062	0.190
GNE	0.029	0.396	0.016	0.071
ComE	0.029	0.314	0.018	0.053
GEMSEC	0.023	0.277	0.107	0.263
ReinCom	**0.058**	**0.401**	**0.138**	**0.281**

Table 5: Self-comparisons on two datasets.

	Aminer		Wiki-Vote	
	Modularity	AUC	Modularity	AUC
Non-hierarchical	0.703	0.950	0.326	0.853
Random	0.719	0.957	0.295	0.846
w/o. Hyperbolic	0.728	0.948	0.295	0.870
ReinCom	**0.759**	**0.960**	**0.327**	**0.884**

Table 6: Inference time of different methods.

	Wiki-Vote	Deezer-RO	Deezer-HR
Nodes	3513	11847	42586
Edges	95028	105844	935138
MNMF	4min	39min	N/A
vGraph	240min	N/A	N/A
ReinCom	45min	50min	500min

Conclusion

- We present the first deep learning based framework on hierarchical community detection
- Empirical results on four real-world complex networks validate the effectiveness of our framework compared with existing heuristic approaches
- Besides:
  - Online updating for new observations
  - Application to multiple downstream tasks due to the learned embeddings

Future Work

- Optimization for industry-scale networks
- Integrate more candidate operations such as delete and split for the community agent
- Leverage node attributes to further improve the performance
- ……

Thank you for listening!

If you have any questions, please contact us.