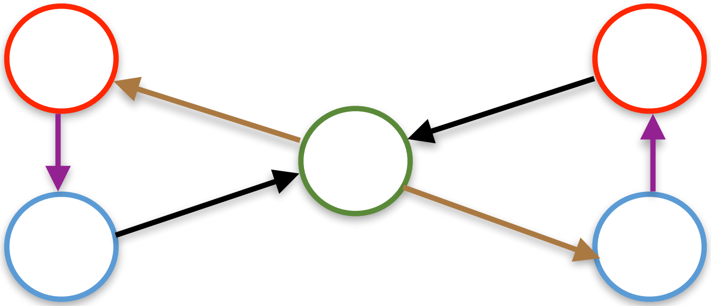
On the Bottleneck of Graph Neural Networks and its Practical Implications

Uri Alon
Technion

Eran Yahav
Technion

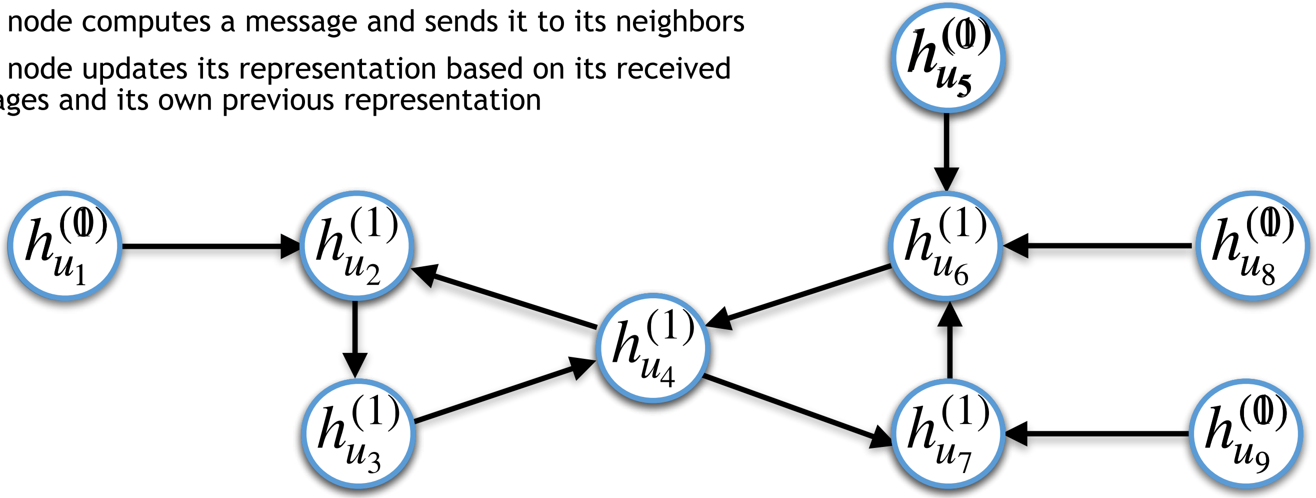
Graph Neural Networks (GNNs)

- Can efficiently learn graph-based data *G=(V, E)*:
  - *V* - Nodes
  - *E* - Typed, directed, edges
- Useful for learning social networks, knowledge graphs, product recommendation, programs
- Very general - can encode any data that can be represented as a graph



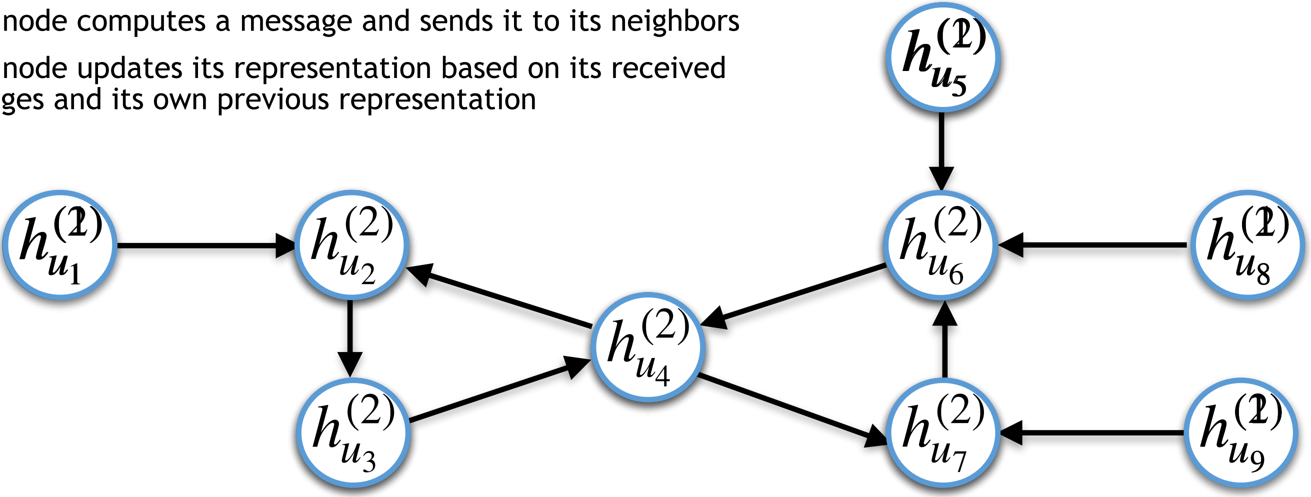
A GNN as a Message Passing Network [Gilmer, ICML'2017]

- Initial representations are embeddings or features
- At every message passing step (=layer):
  - Every node computes a message and sends it to its neighbors
  - Every node updates its representation based on its received messages and its own previous representation



A GNN as a Message Passing Network [Gilmer, ICML’2017]

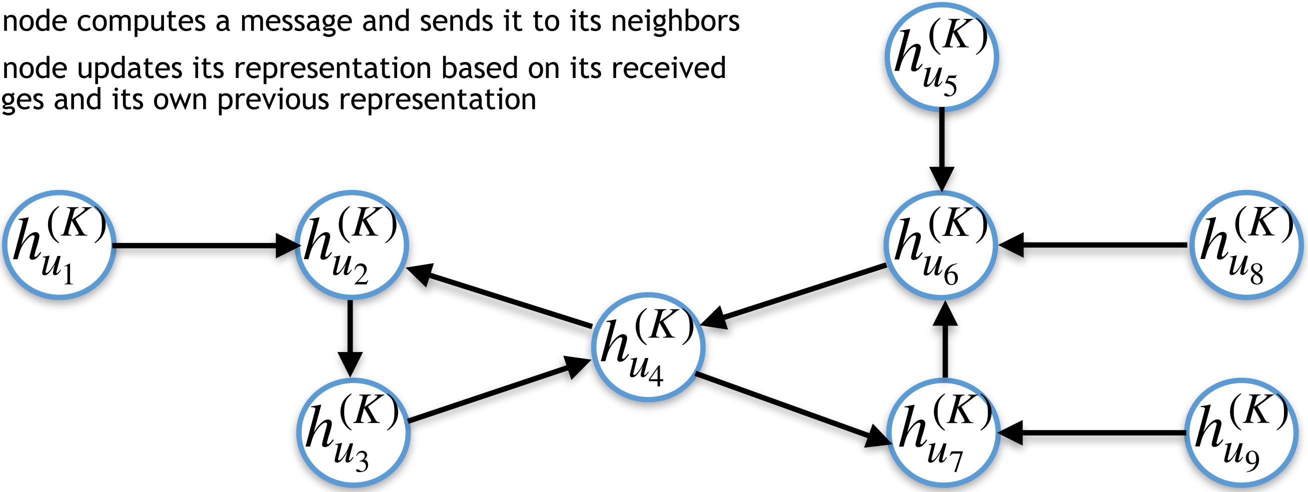
- Initial representations are embeddings or features
- At every message passing step (=layer):
    - Every node computes a message and sends it to its neighbors
    - Every node updates its representation based on its received messages and its own previous representation



$h_{u_5}^{(1)}$ $h_{u_5}^{(2)}$
$h_{u_1}^{(1)}$ $h_{u_1}^{(2)}$
$h_{u_2}^{(2)}$
$h_{u_6}^{(2)}$
$h_{u_8}^{(1)}$ $h_{u_8}^{(2)}$
$h_{u_4}^{(2)}$
$h_{u_3}^{(2)}$
$h_{u_7}^{(2)}$
$h_{u_9}^{(1)}$ $h_{u_9}^{(2)}$

A GNN as a Message Passing Network [Gilmer, ICML’2017]

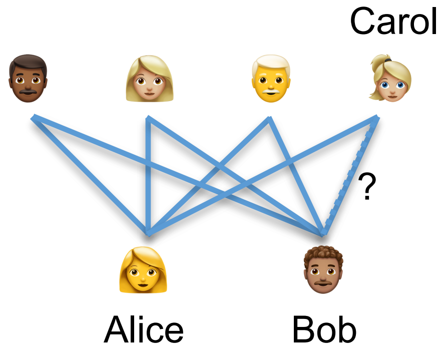
- Initial representations are embeddings or features
- At every message passing step (=layer):
  - Every node computes a message and sends it to its neighbors
  - Every node updates its representation based on its received messages and its own previous representation



- Given $\{h_u^{(K)} \mid u \in V\}$:
  - Node classification, graph classification, link prediction, sequence generation

What are graph neural networks good for?

- GNNs are good for **short-range** tasks:
  - Paper subject classification (Cora/Citeseer/Pubmed, Sen et al., 2008)
  - Friendship/collaboration prediction (Open Graph Benchmark, Hu et al. 2020):



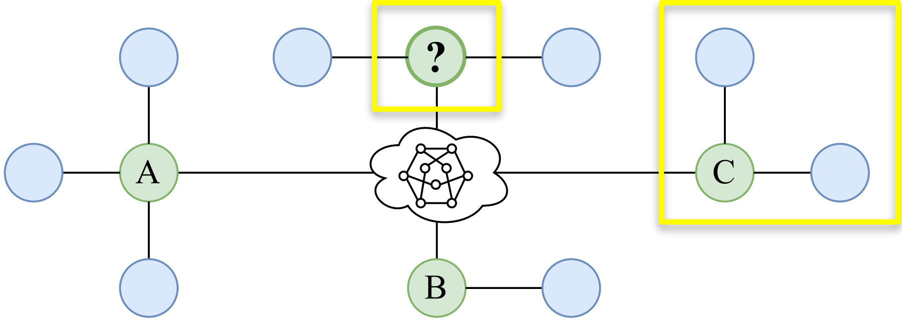
Very local property,

requires only 2-3 message-passing steps

- This work: but not that good for long-range tasks — tasks that require many message-passing steps (~4+)

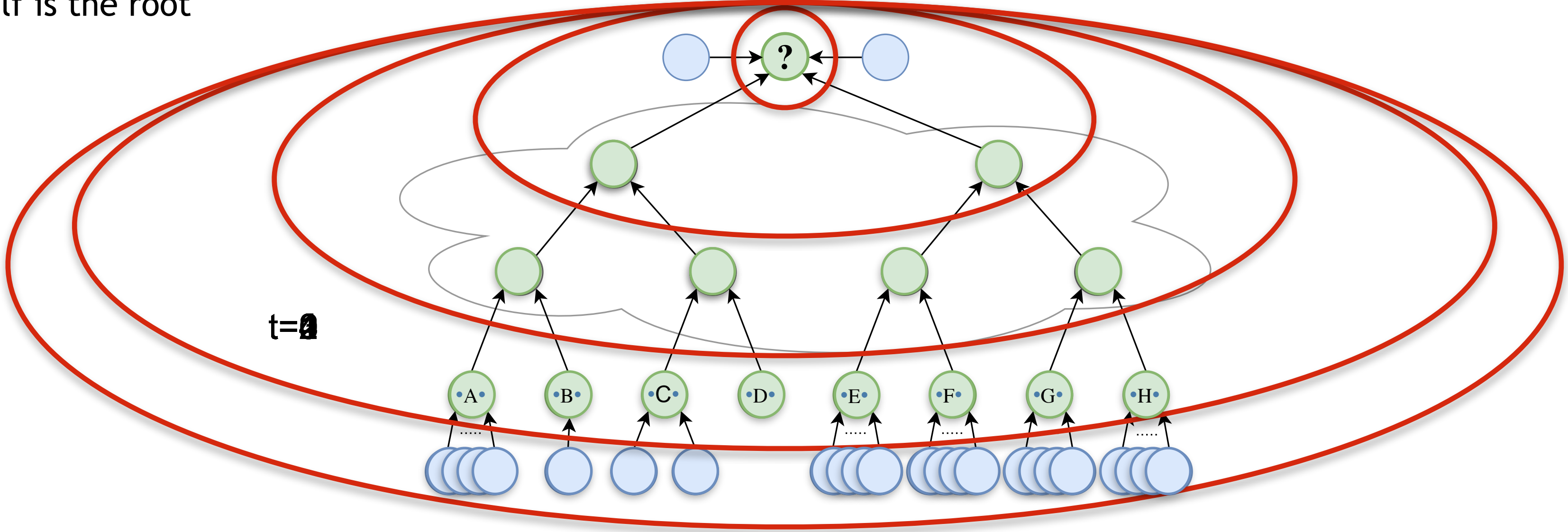
The NeighborsMatch problem

- Assume that we wish to predict a label for the target node
- The correct label is the label of the green node that has the same number of blue neighbors as the target node, in the same graph
  - In this example – **C**



The GNN Bottleneck

From the perspective of the target node, the rest of the graph may look like a tree, where the target node itself is the root



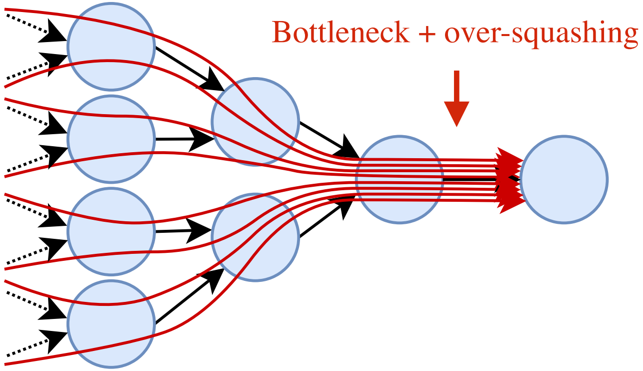
We need: $K > distance\left(C, target\right)$

In this case, we need at least 4 GNN layers for the information to reach the target node

However, the receptive field of the target node grows **exponentially** with the number of layers

Over-squashing

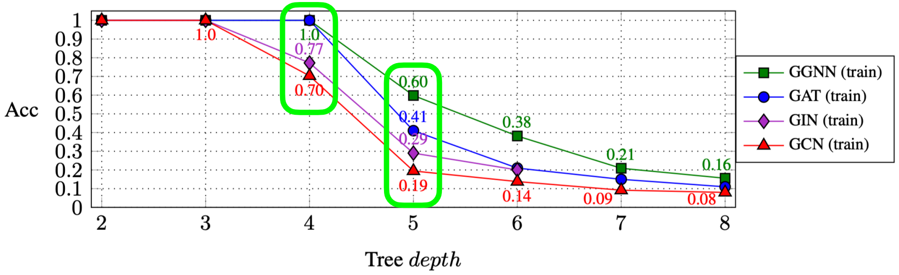
To flow a message to a distance of 4, we need to squash $\mathbf{O}\left(\mathbf{degree}^4\right)$ messages into a single node representation (the representation of the target node).



An exponential amount of information is squashed into a fixed-size vector.

Over-squashing prevents GNNs from fitting the training data

- At depth=4, some GNNs cannot even reach 100% **training** accuracy

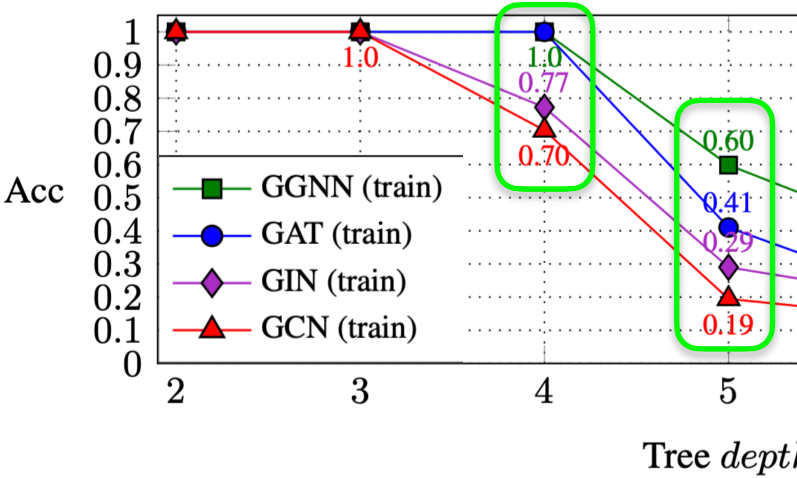


- (In the paper:) combinatorially, to fit the dataset, the dimension d must satisfy: $\left(2^{depth}\right)! < 2^{32 \cdot d}$
  - Such that there will be enough bits to express all different training examples
  - Otherwise, pigeonhole principle: some different examples will result in the same vector representation.

Different GNNs are affected by the bottleneck differently

- GCN and GIN suffer from over-squashing more than GAT and GGNN.

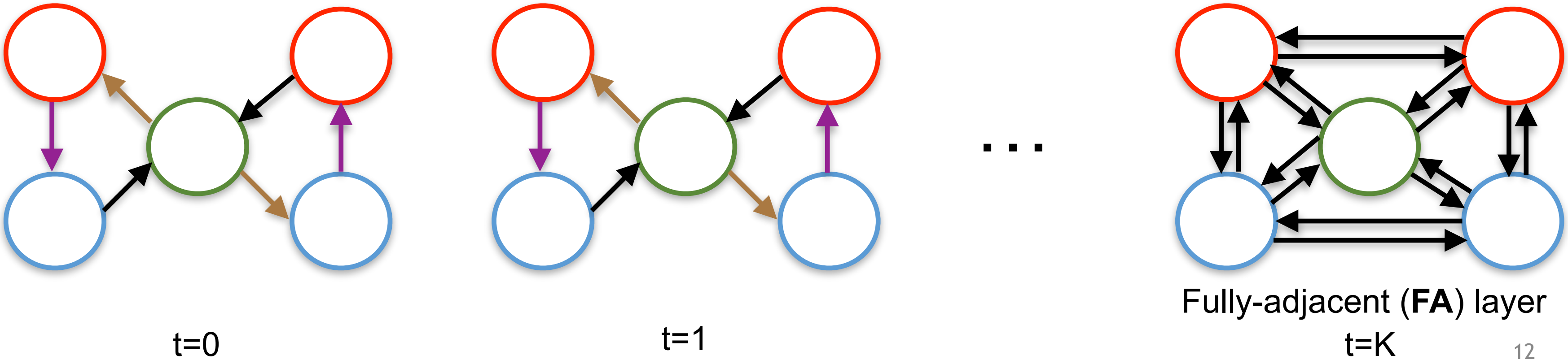
- GCN $$\mathbf{h}_v^{(k)} = ReLU\left(W^{(k)}\left(\frac{1}{c_v}\mathbf{h}_v^{(k-1)} + \boxed{\sum_{u\in\mathcal{N}_v}\frac{1}{c_{v,u}}\mathbf{h}_u^{(k-1)}}\right)\right)$$
- GIN $$\mathbf{h}_v^{(k)} = MLP^{(k)}\left(\left(1+\epsilon^{(k)}\right)\mathbf{h}_v^{(k-1)} + \boxed{\sum_{u\in\mathcal{N}_v}\mathbf{h}_u^{(k-1)}}\right)$$
- GAT $$\mathbf{h}_v^{(k)} = ReLU\left(\boxed{MultiHeadAttention\left(\mathcal{N}_v \mid \mathbf{h}_v^{(k-1)}\right)}\right)$$
- GGNN $$\mathbf{h}_v^{(k)} = \boxed{GRU}\left(\mathbf{h}_v^{(k-1)}, \sum_{u\in\mathcal{N}_v} W_{neighbor}\mathbf{h}_u^{(k-1)}\right)$$



More - in the paper…

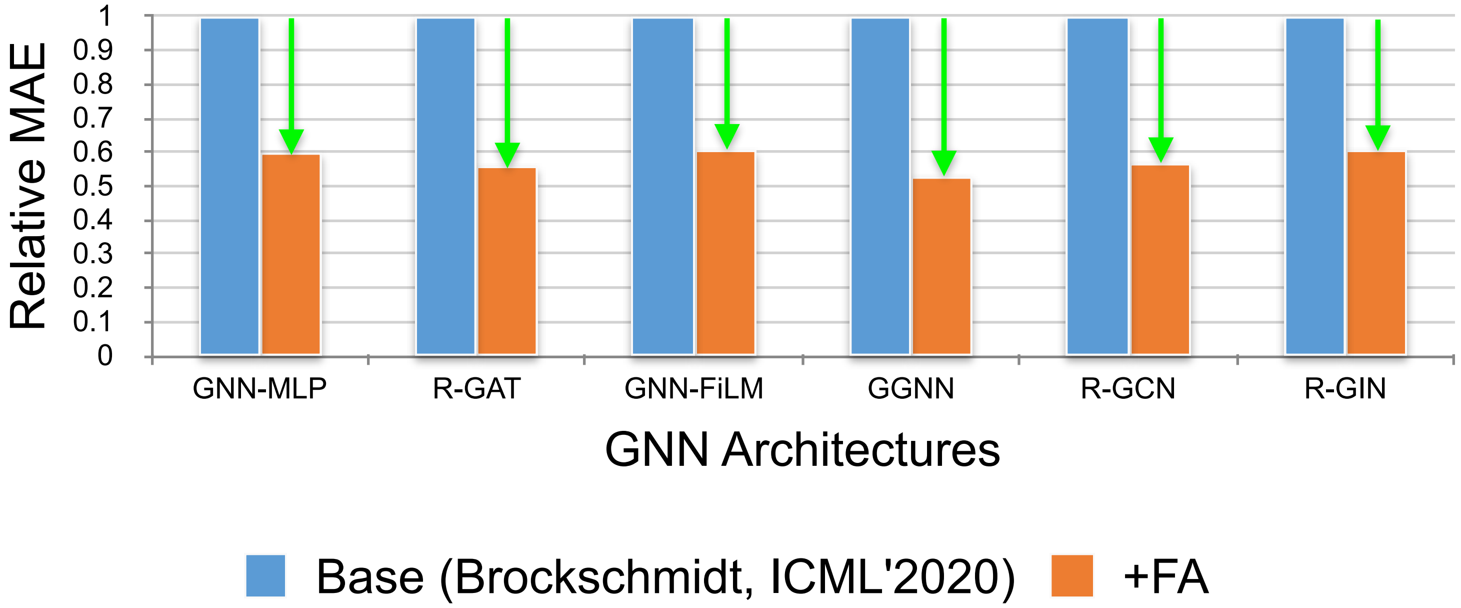
Does the bottleneck affect real-life models?

- Off-the-shelf, state-of-the-art models, trained by others
- To break the bottleneck:
  - We (modified the original implementations and) made the last GNN layer fully-adjacent (**FA**) - every node has an edge to every other node
  - Re-trained without adding weights, without **any** hyperparameter tuning
- The most trivial idea, just to show that the bottleneck affects SoTA models



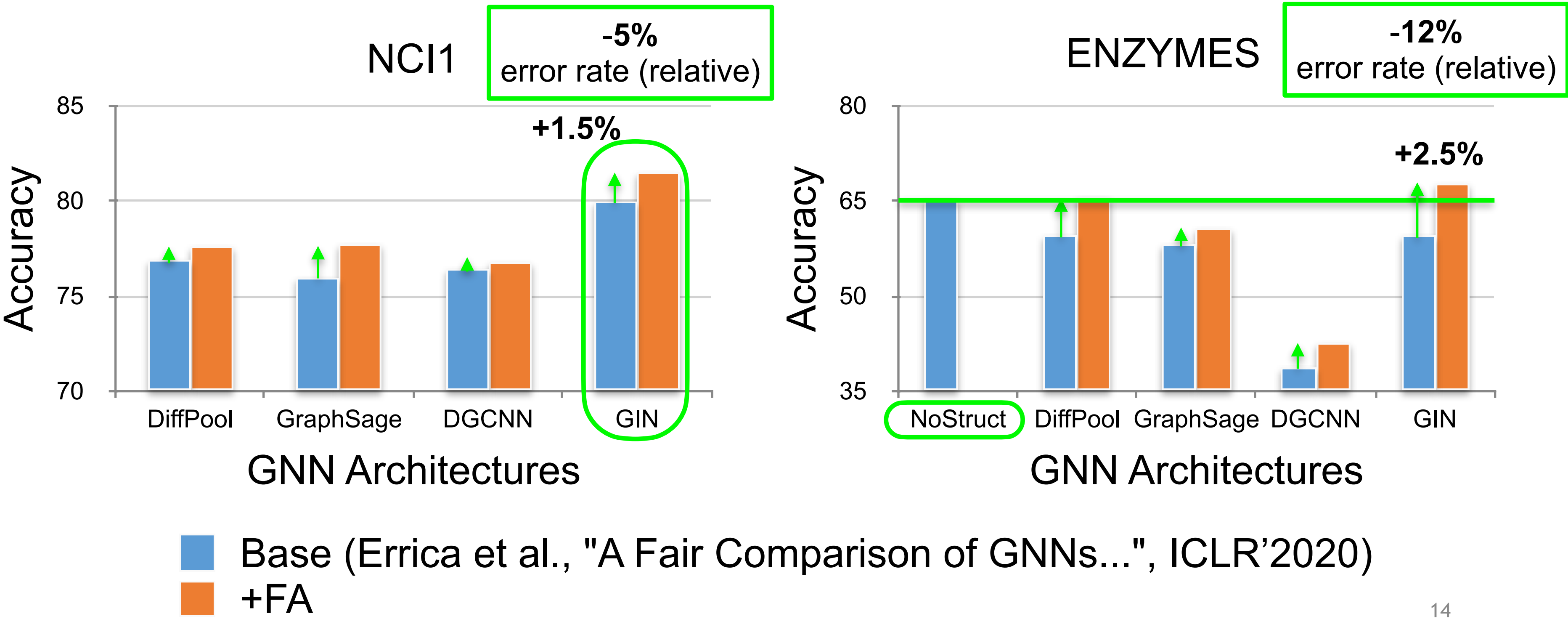
QM9 Dataset (molecules regression)

(mean absolute error, lower = better)



-40% error

Biological Datasets
(accuracy, higher = better)



-5% error rate (relative)

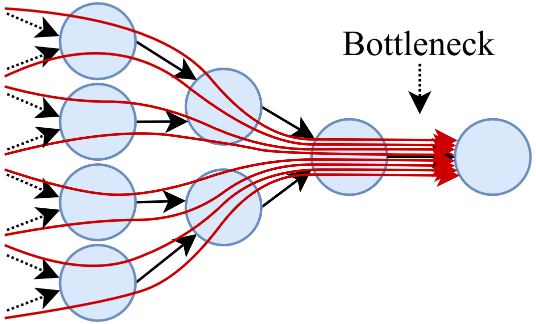
+1.5%

-12% error rate (relative)

+2.5%

Summary

- To pass long-range messages - we need many GNN layers
- A node's receptive field grows **exponentially** with the number of layers
  - ➡ Leads to a **bottleneck** and **over-squashing**
- **GCN** and **GIN** suffer from over-squashing **more** than others
- SoTA models can be **improved** by simply considering the bottleneck
- Still looking for better solutions



urialon@cs.technion.ac.il

http://urialon.ml