

# Graph Neural Networks (GNNs)

Gabriele Ciravegna

\*Part of these slides are to be attributed to Petar Veličković which I thank for sharing his presentation

# Talk roadmap

1. What are Graphs? What are they used to?
2. Learning with GNN: A historical Perspective
3. Derive GNNs from first principles
4. Learning on Graph
5. The Message Passing framework
6. GNN Limitations
7. Novel trends in GNN

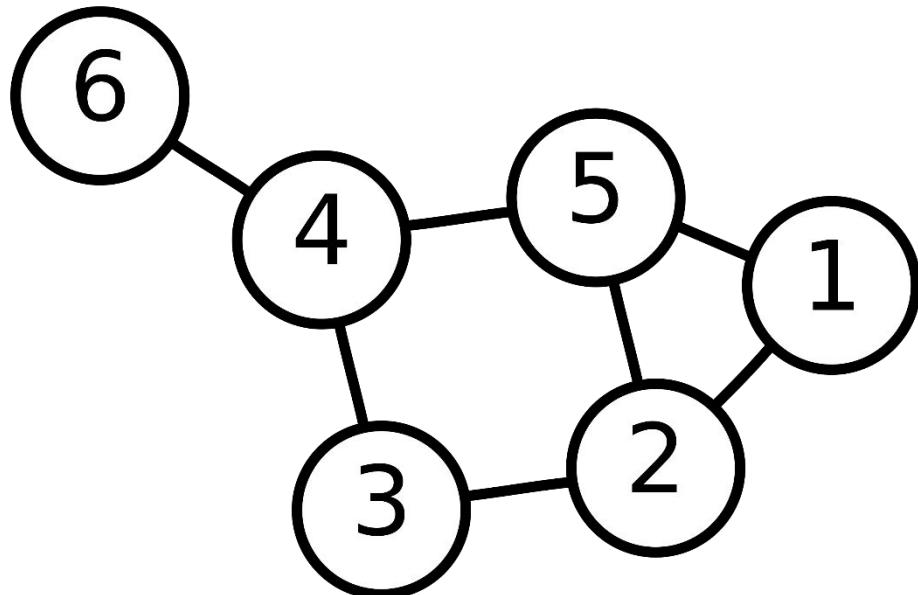
# Fantastic GNNs in the Wild

---

# What is a Graph?

---

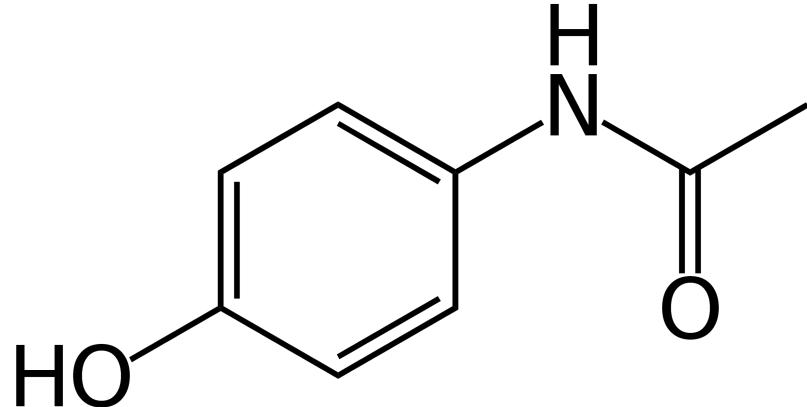
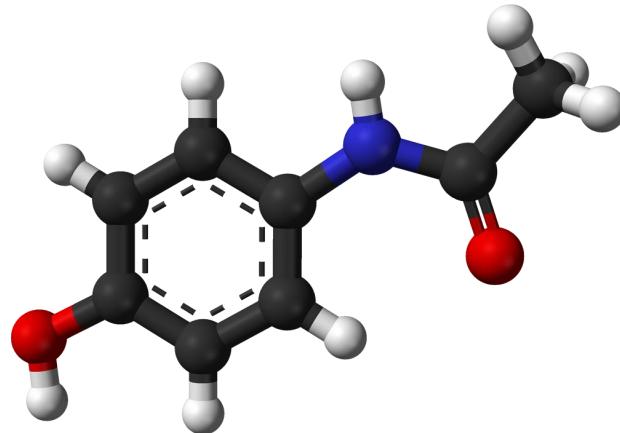
- Mathematical structures used to model pairwise relations between objects
- Graphs  $G = (V, E)$  are made up of:
  - Vertices  $V$  (also called *nodes* or *points*)
  - Edges  $E$  (also called *links* or *lines*) which connects the vertices



# Molecules are graphs!

---

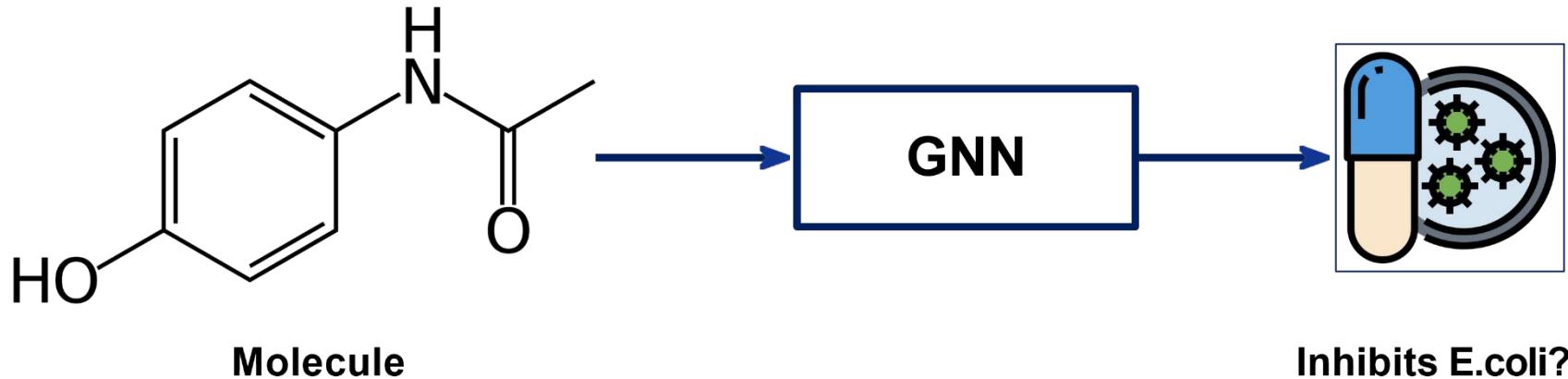
- A very natural way to represent molecules is as a **graph**
  - **Atoms** as nodes, **bonds** as edges
  - Features such as **atom type**, **charge**, **bond type**...



# GNNs for molecule classification

Interesting task, is a molecule a potent **drug**?

- *Binary classification* on whether the drug will inhibit a bacteria
- Train on a **curated dataset** for compounds where response is known



# Follow-up study

Once trained, the model can be applied to *any* molecule

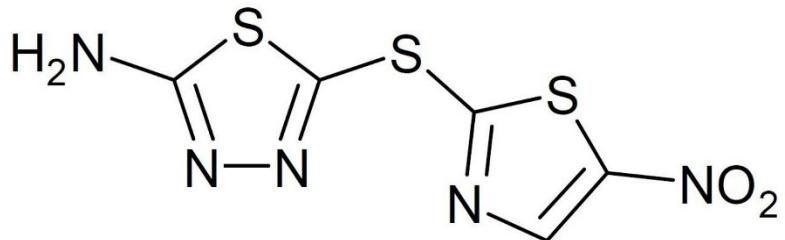
Execute on a large dataset of known candidate molecules

Select the ~*top-100* candidates from your GNN model

Chemists thoroughly investigate



Discover a previously overlooked compound that is a **highly potent** antibiotic!



Halicin

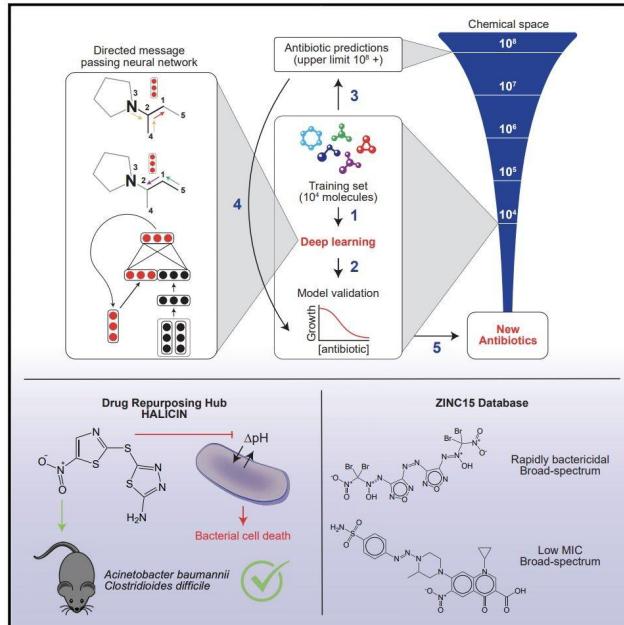
...Achieve wide  
acclaim!

Arguably the most popularised  
**success story** of graph neural  
networks!

Cell

## A Deep Learning Approach to Antibiotic Discovery

### Graphical Abstract



### Authors

Jonathan M. Stokes, Kevin Yang,  
Kyle Swanson, ..., Tommi S. Jaakkola,  
Regina Barzilay, James J. Collins

### Correspondence

regina@csail.mit.edu (R.B.),  
jimjc@mit.edu (J.J.C.)

### In Brief

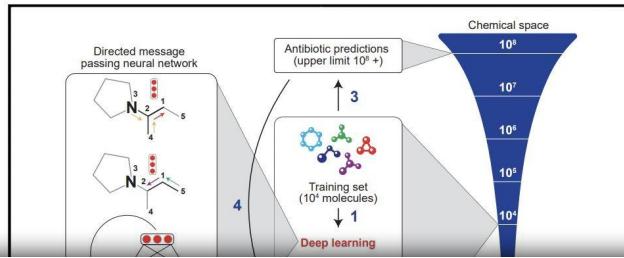
A trained deep neural network predicts antibiotic activity in molecules that are structurally different from known antibiotics, among which Halicin exhibits efficacy against broad-spectrum bacterial infections in mice.

...Achieve wide  
acclaim!

# Cell

## A Deep Learning Approach to Antibiotic Discovery

### Graphical Abstract



### Authors

Jonathan M. Stokes, Kevin Yang,  
Kyle Swanson, ..., Tommi S. Jaakkola,  
Regina Barzilay, James J. Collins

### Correspondence

regina@csail.mit.edu (R.B.),  
jimjc@mit.edu (J.J.C.)

### In Brief

Subscribe

# nature

NEWS · 20 FEBRUARY 2020

## Powerful antibiotics discovered using AI

Machine learning spots molecules that work even against ‘untreatable’ strains of bacteria.

Arguably the most popularised  
**success story** of graph neural  
networks!



# ...Achieve wide acclaim!

Arguably the most popularised **success story** of graph neural networks!

Cell

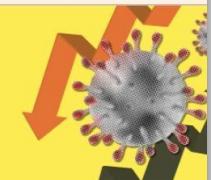
AI IN USE

## FINANCIAL TIMES

S COMPANIES TECH MARKETS GRAPHICS OPINION WORK & CAREERS LIFE & ARTS HOW TO SPEND IT

### CORONAVIRUS BUSINESS UPDATE

Get 30 days' complimentary access to our Coronavirus Business Update newsletter



#### Artificial intelligence

##### Robotics



'Death of the office' homeworking claims exaggerated



Anti-social robots help increase social distancing

Artificial intelligence

+ Add to myFT

NEWS

PODCAST

Mac

## AI discovers antibiotics to treat drug-resistant diseases

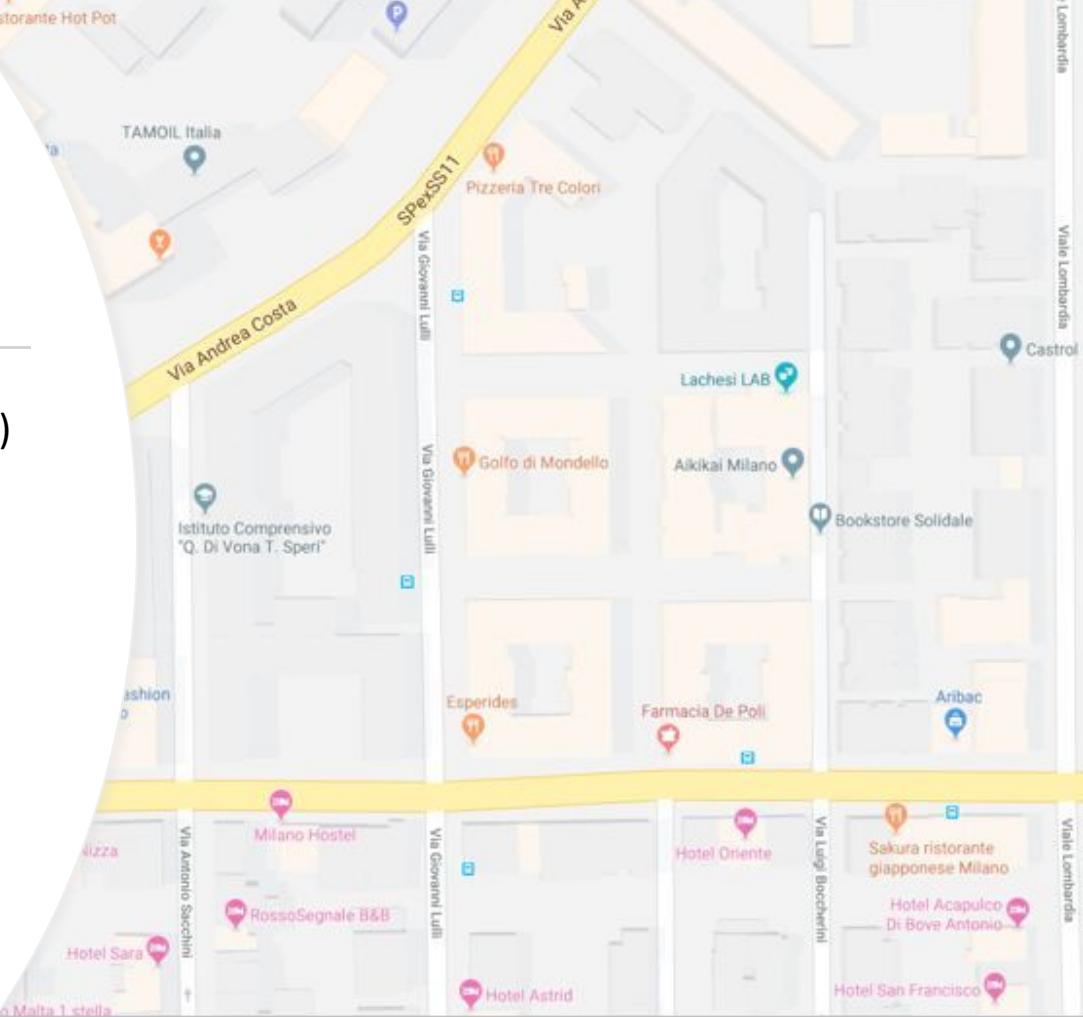
Machine learning uncovers potent new drug able to kill 35 powerful bacteria

of

bacteria.

# Traffic maps are graphs!

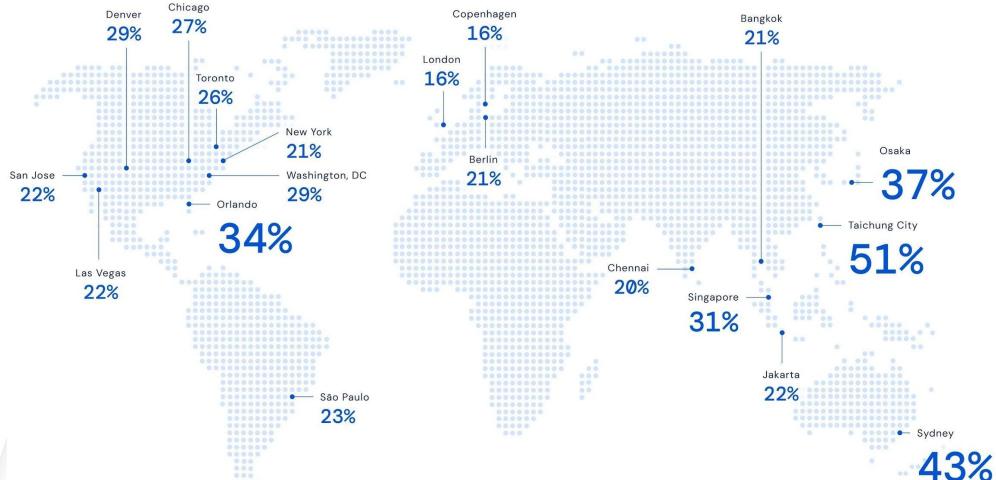
- Transportation maps (e.g. *Google Maps*) can be naturally modelled as **graphs**
- Nodes could be **intersections**, and edges could be **roads**
- Relevant **features**: road *length*, *current average speed*, *historical speed*



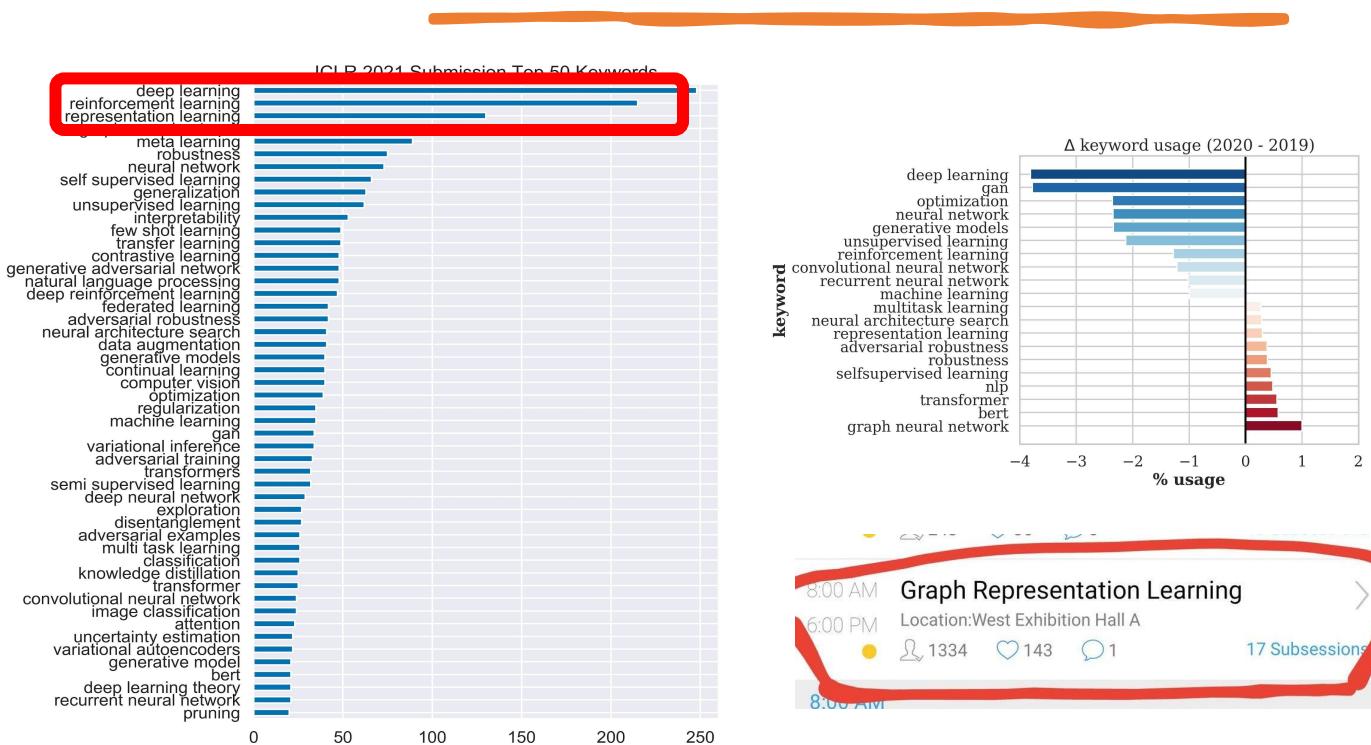
# DeepMind's ETA Prediction using GNNs in Google Maps

Run GNN on  
**supersegment** graph to  
estimate **time of arrival**  
(ETA) (*graph regression*).

Already **deployed** in  
several major cities,  
significantly improving  
ETA outcomes!



# GNNs are a very hot research topic



**GNNs are  
currently  
experiencing their  
“ImageNet”  
moment**

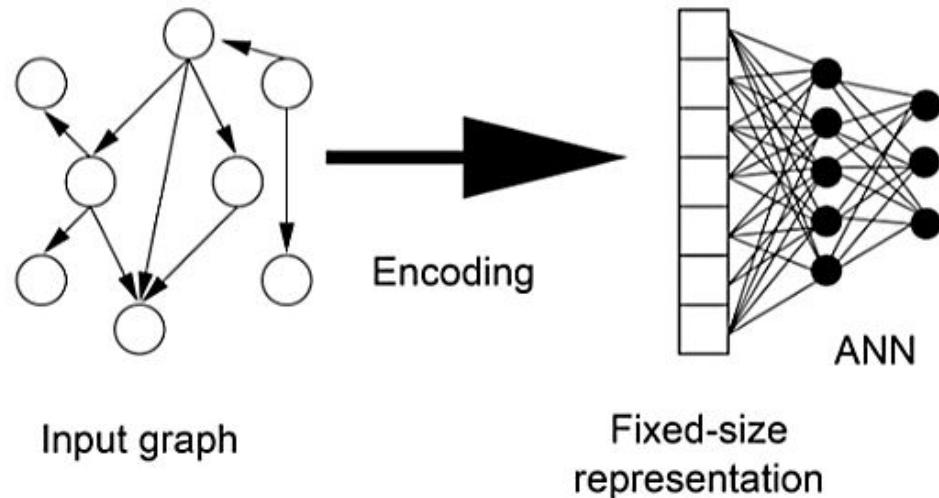
# Learning with Graphs – An Historical Perspective



# Traditional ML approaches

---

- ML approaches assume to deal with **flat data**
- *Pre-processing step*, extracting structural information encodings
- Flat representations relying on summary graph statistics, kernel functions, graph traversals, etc.
- **Cons:**
  - loosing useful information,
  - not able to adapt during learning



# GNN Advantage

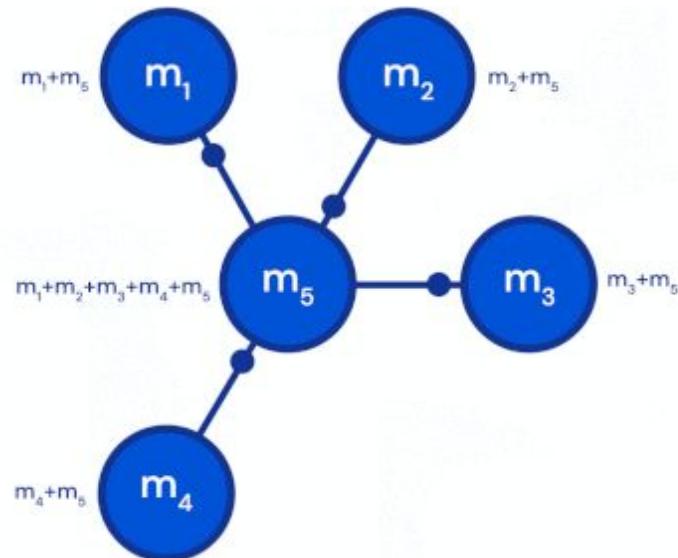
---

No need of a preprocessing embedding step

No limitation on the graph type

Neural networks exploited to learn how to encode nodes of a graph for a given task

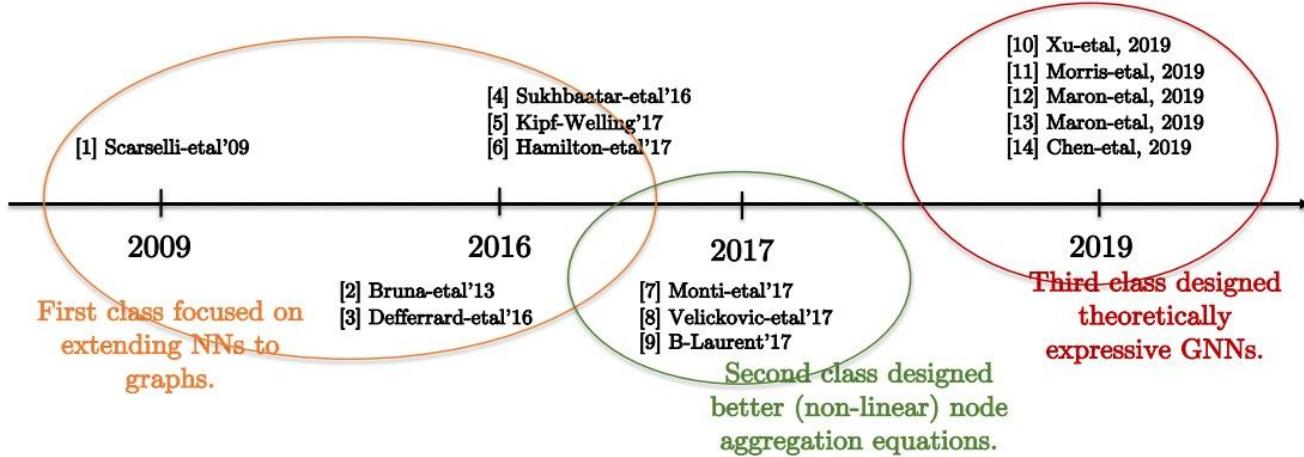
Consider information local to each node and the whole graph topology



# Where did GNN comes from?

- Early forms can be traced to the **early 1990s**, often involving DAG structures.
  - Backpropagation through structure (Goller & Küchler, ICNN'96)
  - Adaptive structure processing (Sperduti & Starita, TNN'97; Frasconi *et al.*, TNN'98)
- First proper treatment of **generic** graph structure processing happens in the 2000s:
  - The **GNN framework** (Gori *et al.*, IJCNN'05; Scarselli *et al.*, TNN'08)
  - The **NN4G framework** (Micheli, TNN'09)
- The GNN model of Gori, Scarselli *et al.* used primarily *recurrent-style* updates
  - Problems in the optimization of the message diffusion process

# GNN - Timeline

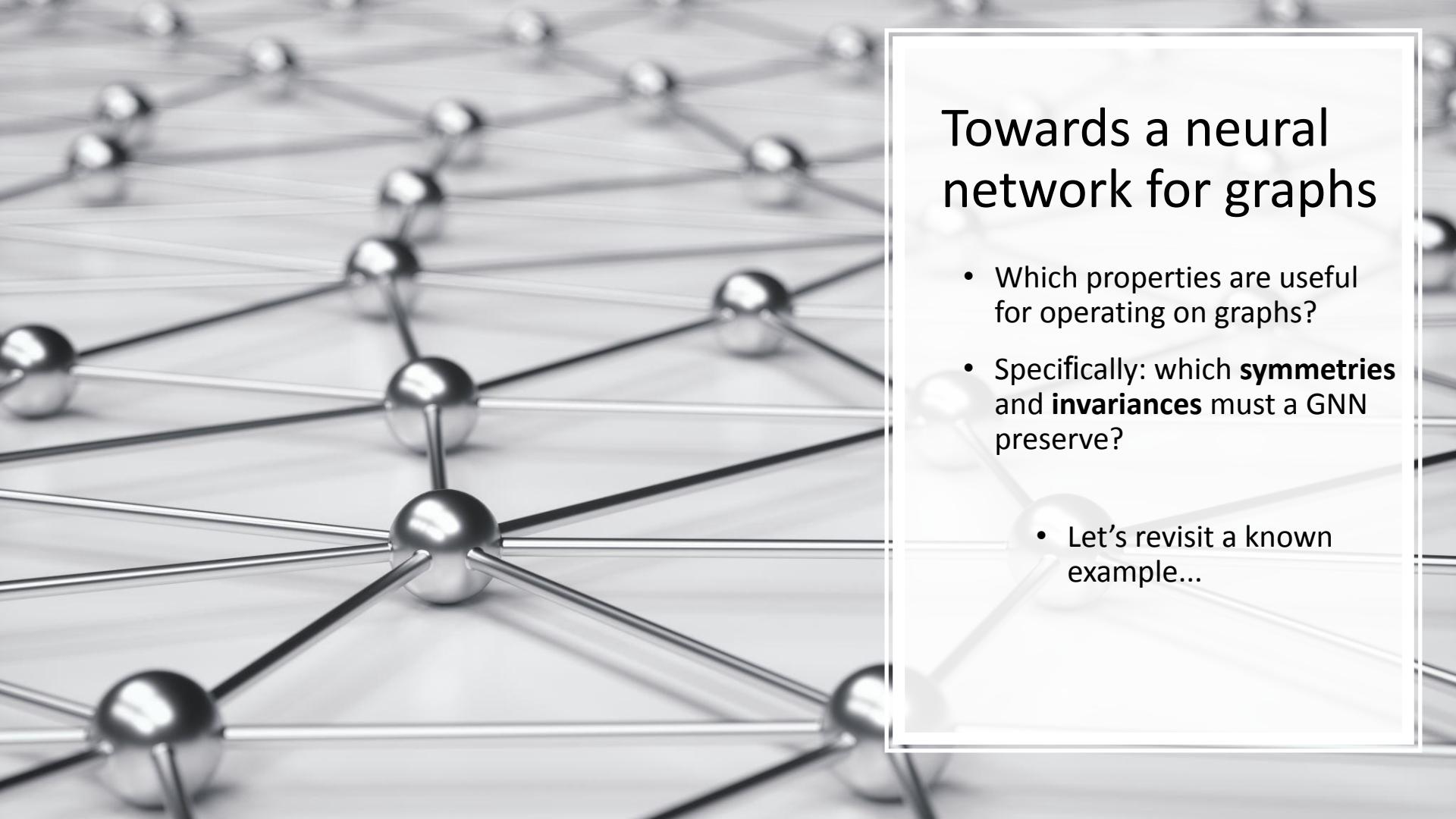


- [1] Scarselli, Gori, Tsoi, Hagenbuchner, Monfardini, The Graph Neural Network Model, 2009
- [2] Bruna, Zaremba, Szlam, LeCun, Spectral networks and locally connected networks on graphs, 2013
- [3] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016
- [4] Sukhbaatar, Szlam, Fergus, Learning multiagent communication with backpropagation, 2016
- [5] Kipf, Welling, Semi-supervised classification with graph convolutional networks, 2017
- [6] Hamilton, Ying, Leskovec, Inductive representation learning on large graphs, 2017
- [7] Monti, Boscaini, Masci, Rodola, Svoboda, Bronstein, Geometric deep learning on graphs using mixture model cnns, 2017
- [8] Velickovic, Cucurull, Casanova, Romero, Lio, Bengio, Graph attention networks, 2017
- [9] Bresson, Laurent, Residual gated graph convnets, 2017
- [10] Xu, Hu, Leskovec, Jegelka, How powerful are graph neural networks?, 2019
- [11] Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe, Weisfeiler and leman go neural: Higher-order graph networks, 2019
- [12] Maron, Ben-Hamu, Shamir, Lipman, Invariant and equivariant graph networks, 2019
- [13] Maron, Ben-Hamu, Serviansky, Lipman, Provably powerful graph networks, 2019
- [14] Chen, Villar, Chen, Bruna, On the equivalence graph isomorphism testing and function approximation with gnnns, 2019

Apologize for not citing more works (4000+ GNN papers).

# Towards GNNs from first principles

---



# Towards a neural network for graphs

- Which properties are useful for operating on graphs?
- Specifically: which **symmetries** and **invariances** must a GNN preserve?
- Let's revisit a known example...

# Convolution on images

The diagram illustrates the convolution operation  $I * K$  on an input image  $I$  with a kernel  $K$ .

**Input Image  $I$ :**

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

**Kernel  $K$ :**

1	0	1
0	1	0
1	0	1

**Result  $I * K$ :**

1	4	3	4	1
1	2	4	3	3
1	2	3	4	1
1	3	3	1	1
3	3	1	1	0

Detailed description: The diagram shows the convolution process between a 7x7 input image  $I$  and a 3x3 kernel  $K$ . The result is a 5x5 output matrix  $I * K$ . The input  $I$  has values 0, 1, and 2. The kernel  $K$  has values 1, 0, and 1. The output  $I * K$  has values 1, 4, 3, 4, 1; 1, 2, 4, 3, 3; 1, 2, 3, 4, 1; 1, 3, 3, 1, 1; and 3, 3, 1, 1, 0. Dotted lines show the receptive fields of the top-left cell in the output matrix, which is highlighted with a green box.

# Convolution on images

The diagram illustrates the convolution operation  $I * K$  on image  $I$ . The input image  $I$  is a 7x7 matrix with values:

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

The kernel  $K$  is a 3x3 matrix with values:

1	0	1
0	1	0
1	0	1

The result of the convolution  $I * K$  is a 5x5 matrix with values:

1	4	3	4	1
1	2	4	3	3
1	2	3	4	1
1	3	3	1	1
3	3	1	1	0

Dotted arrows indicate the receptive fields of the output units. The highlighted green square in the result matrix indicates the center of the kernel's receptive field.

# Convolution on images

The diagram illustrates the convolution operation  $I * K$  on an input image  $I$  using a kernel  $K$ .

**Input Image  $I$ :**

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	1
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

**Kernel  $K$ :**

1	0	1
0	1	0
1	0	1

**Result  $I * K$ :**

1	4	3	4	1
1	2	4	3	3
1	2	3	4	1
1	3	3	1	1
3	3	1	1	0

Detailed description: The diagram shows the convolution process between a 7x7 input image  $I$  and a 3x3 kernel  $K$ . The result is a 5x5 output image  $I * K$ . The input  $I$  has values 0, 1, and 2. The kernel  $K$  has values 1, 0, and 1. The output  $I * K$  has values 1, 2, 3, 4, and 5. The diagram uses colored boxes and dashed lines to highlight the receptive fields and the calculation of each output unit.

# Convolutional Neural Network Invariances

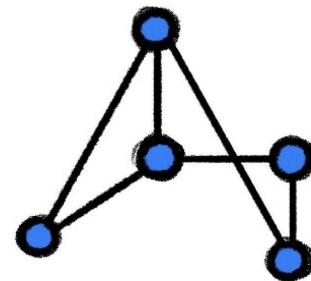
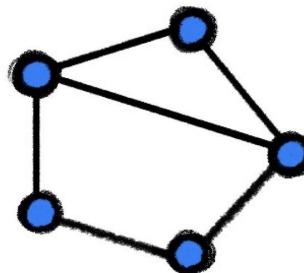


- Convolutional neural nets respect **translational invariance**
- Patterns are interesting irrespective of *where* they are in the image
- **Locality:** neighbouring pixels relate much more strongly than distant ones
- What about **arbitrary** graphs?

# Isomorphism-preserving transformation

---

- The nodes of a graph are not assumed to be in any order
- We would like to get the same results for two isomorphic graphs
- To see how to enforce this, we will define new terms...



# Learning on sets: Setup

- For now, assume the graph **has no edges** (e.g. *set* of nodes,  $V$ ).
- Let  $\mathbf{x}_i \in \mathbb{R}^k$  be the features of node  $i$ .
- We can stack them into a node feature matrix  $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top$  of shape  $n \times k$ :
- That is, the  $i$ th row of  $\mathbf{X}$  corresponds to  $\mathbf{x}_i$

Note that, by doing so, we have specified a **node ordering!**

- We would like the result of any neural networks to not depend on this.

# Permutations matrices

Permutations are the operations that **change** the node order

- e.g., a permutation  $(2, 4, 1, 3)$  means  $\mathbf{y}_1 \leftarrow \mathbf{x}_2, \mathbf{y}_2 \leftarrow \mathbf{x}_4, \mathbf{y}_3 \leftarrow \mathbf{x}_1, \mathbf{y}_4 \leftarrow \mathbf{x}_3$
- there are  $n!$  of them

In linear algebra, each permutation defines an  $n \times n$  **matrix**

- Such matrices are called **permutation matrices**
- They have exactly one 1 in every row and column, and zeros everywhere else
- Their effect when left-multiplied is to permute the rows of  $\mathbf{X}$ , like so:

$$\mathbf{P}_{(2,4,1,3)} \mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \text{---} & \mathbf{x}_1 & \text{---} \\ \text{---} & \mathbf{x}_2 & \text{---} \\ \text{---} & \mathbf{x}_3 & \text{---} \\ \text{---} & \mathbf{x}_4 & \text{---} \end{bmatrix} = \begin{bmatrix} \text{---} & \mathbf{x}_2 & \text{---} \\ \text{---} & \mathbf{x}_4 & \text{---} \\ \text{---} & \mathbf{x}_1 & \text{---} \\ \text{---} & \mathbf{x}_3 & \text{---} \end{bmatrix}$$

# Permutation *invariance* & *equivariance*

## Permutation *invariance*:

- We want to design functions  $f(\mathbf{X})$  over sets that will not depend on the order of the input

$$f(\mathbf{P}\mathbf{X}) = f(\mathbf{X})$$

- Example of permutation invariant operation: Sum, Avg, Max
- Good models to obtain *set-level* outputs

## Permutation *equivariant*:

- We want to still be able to **identify** node outputs, which a permutation-invariant aggregator would destroy!

$$f(\mathbf{P}\mathbf{X}) = \mathbf{P}f(\mathbf{X})$$

- if we permute the nodes, it doesn't matter if we do it **before** or **after** the function

# General *blueprint* for learning on sets

- The Equivariant set functions transform each node input  $\mathbf{x}_i$  into a *latent* vector  $\mathbf{h}_i$  (embedding):

$$\mathbf{h}_i = \psi(\mathbf{x}_i)$$

- $\psi$  is a function **applied in isolation to every node.**
- Stacking  $\mathbf{h}_i$  yields  $\mathbf{H} = f(\mathbf{X})$ .

- General *blueprint*: stacking **equivariant** function(s), potentially aggregated with an **invariant** operator -- yields many useful functions on sets!

$$f(\mathbf{X}) = \phi \left( \bigoplus_{i \in \mathcal{V}} \psi(\mathbf{x}_i) \right)$$

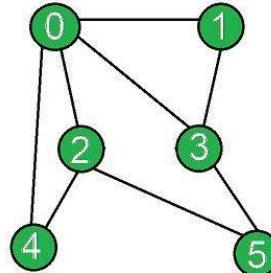
- Here,  $\oplus$  is a permutation-invariant **aggregator** (such as sum, avg or max).

# Learning on graphs

- Now we augment the set of nodes with **edges** between them:  $E \subseteq V \times V$
- We represent these edges with an **adjacency matrix**,  $A$ :

$$a_{ij} = \begin{cases} 1 & (i, j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

- Further additions (e.g. *edge features*) are possible but **ignored** for simplicity here.



	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	0	1	1
3	1	1	0	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

# Permutation invariance and equivariance on graphs

- Node permutations now also valid on the **edges**
- We need to appropriately permute both **rows** and **columns** of  $\mathbf{A}$ 
  - When permutation  $\mathbf{P}$ , on a matrix it amounts to  $\mathbf{P} \mathbf{A} \mathbf{P}^T$
- We update definitions of suitable functions  $f(\mathbf{X}, \mathbf{A})$  over graphs:
- **Invariance:** 
$$f(\mathbf{P}\mathbf{X}, \mathbf{P}\mathbf{A}\mathbf{P}^T) = f(\mathbf{X}, \mathbf{A})$$
- **Equivariance:** 
$$f(\mathbf{P}\mathbf{X}, \mathbf{P}\mathbf{A}\mathbf{P}^T) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$$

# Locality on graphs: Neighborhoods!

- On **sets**, we enforced equivariance applying functions to node **in isolation**
- **Graphs** give us a broader context: a node's **neighbourhood**
  - For a node  $i$ , its (1-hop) neighbourhood is defined as follows:

$$\mathcal{N}_i = \{j : (i, j) \in \mathcal{E} \vee (j, i) \in \mathcal{E}\}$$

**N.B.** we consider *undirected* edges, and often we assume  $i \in \mathcal{N}_i$

- We extract and aggregate the *multiset* of **features** in the neighbourhood

$$\mathbf{X}_{\mathcal{N}_i} = \{\{\mathbf{x}_j : j \in \mathcal{N}_i\}\}$$

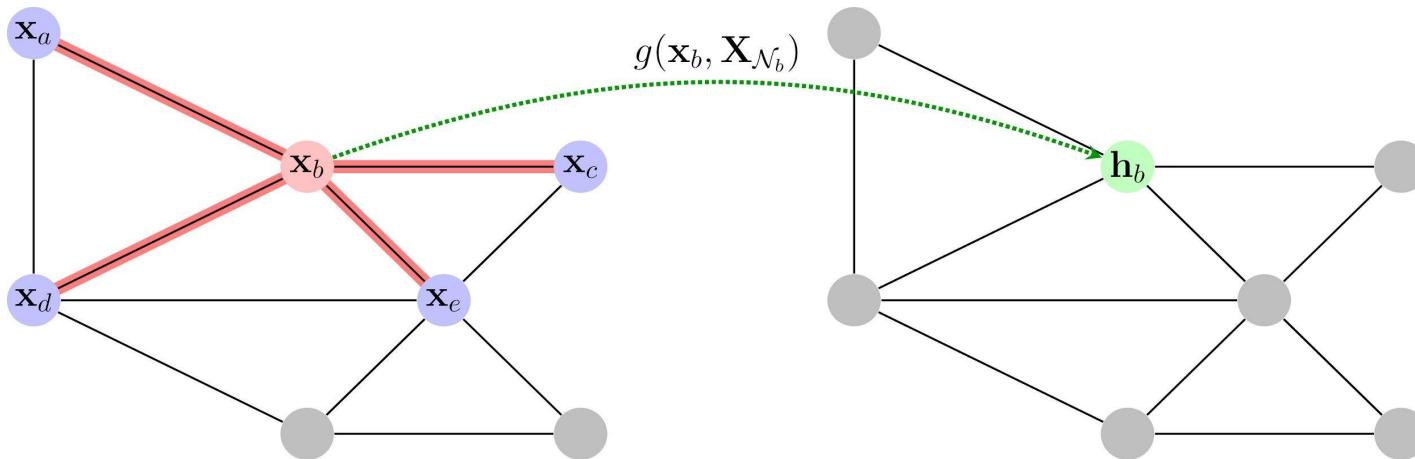
# A recipe for graph neural networks

- We construct permutation equivariant functions  $f(\mathbf{X}, \mathbf{A})$  by applying the local function,  $g$ , over *all* neighbourhoods:

$$f(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} \text{---} & g(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & \text{---} \\ \text{---} & g(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & \text{---} \\ & \vdots & \\ \text{---} & g(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & \text{---} \end{bmatrix}$$

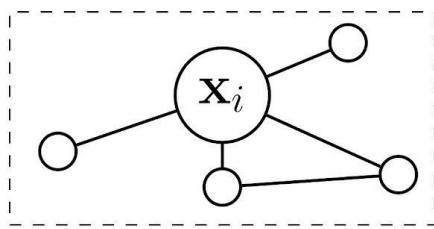
- To ensure equivariance, we need  $g$  to not depend on the **order** of vertices in  $\mathbf{X}_{\mathcal{N}_i}$ 
  - Hence,  $g$  should be permutation **invariant**!

# A recipe for graph neural networks, visualised



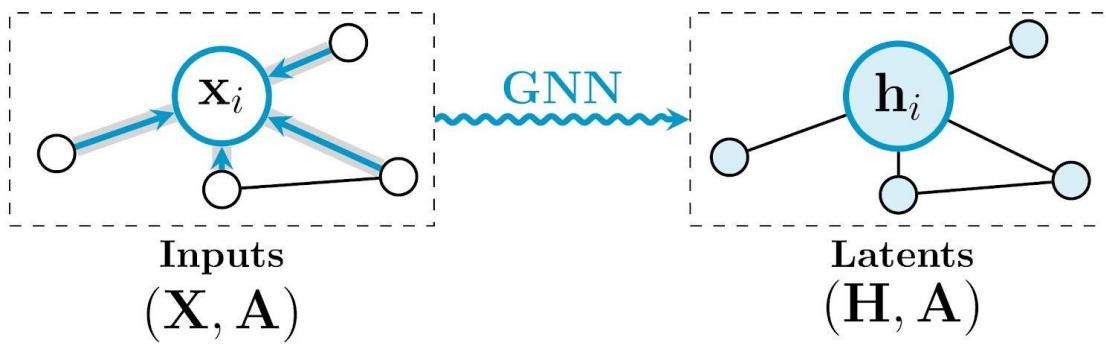
$$\mathbf{X}_{\mathcal{N}_b} = \{\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}\}$$

# How to use GNNs?

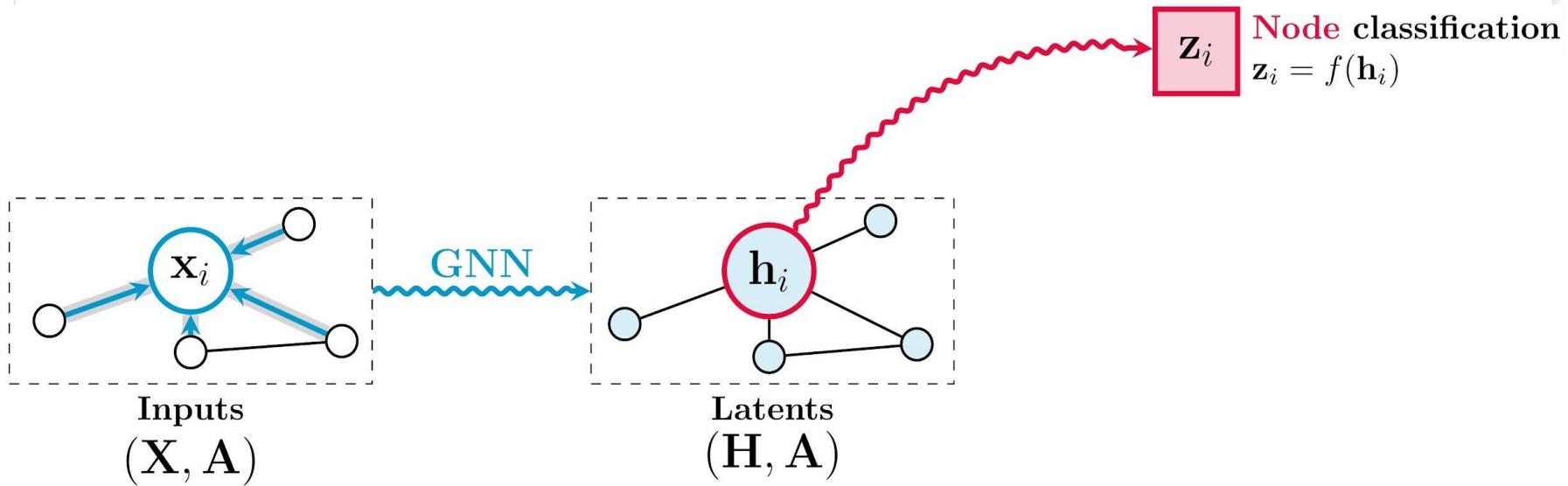


Inputs  
 $(\mathbf{X}, \mathbf{A})$

# How to use GNNs?

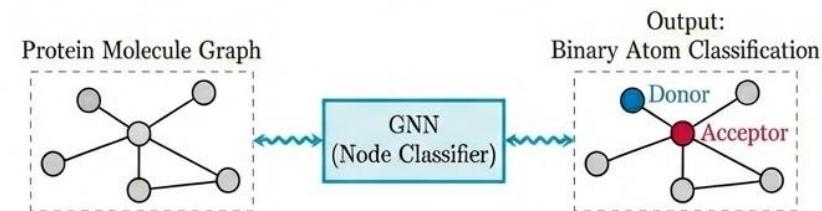


# How to use GNNs?



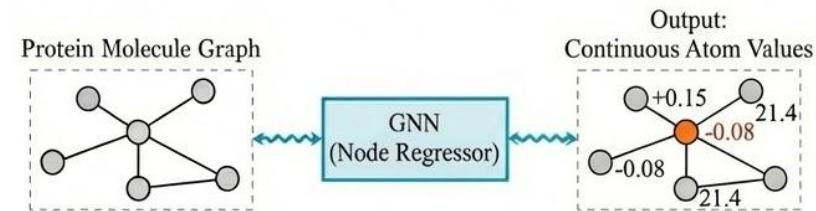
# Node Classification Example

- Predicting a discrete category for a node.
- **Example:** Classifying whether a specific atom (node) in a protein is an ‘acceptor’ or ‘donor’ for hydrogen bonding (Binary Class).

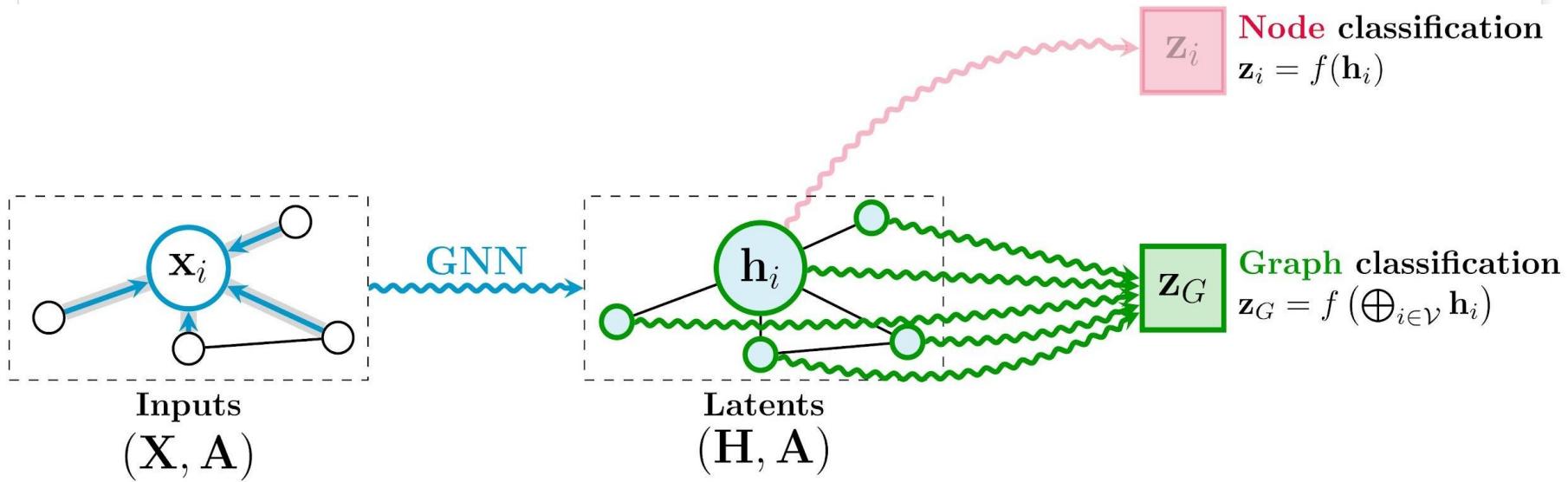


# Node Regression Example

- Predicting a continuous value for a node.
- **Example:** Predicting the precise partial charge or magnetic shielding value of a specific atom within the molecule.

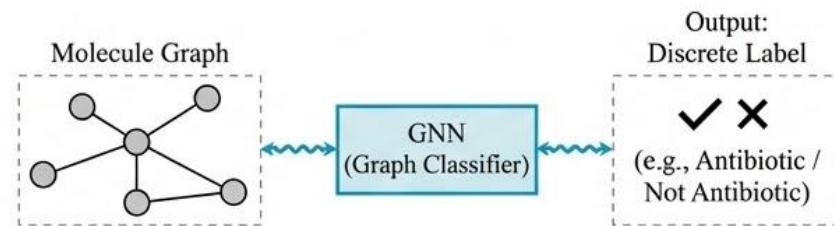


# How to use GNNs?



# Graph Classification Example

- Predicting a discrete label for the whole graph.
- **Example (Antibiotics):** The "Halicin" discovery case study is a prime example.
  - The task was a binary classification on whether the whole molecule (graph) will inhibit E. coli bacteria.

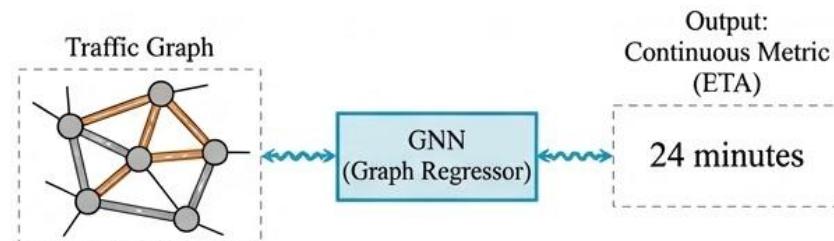


# Graph Regression Example

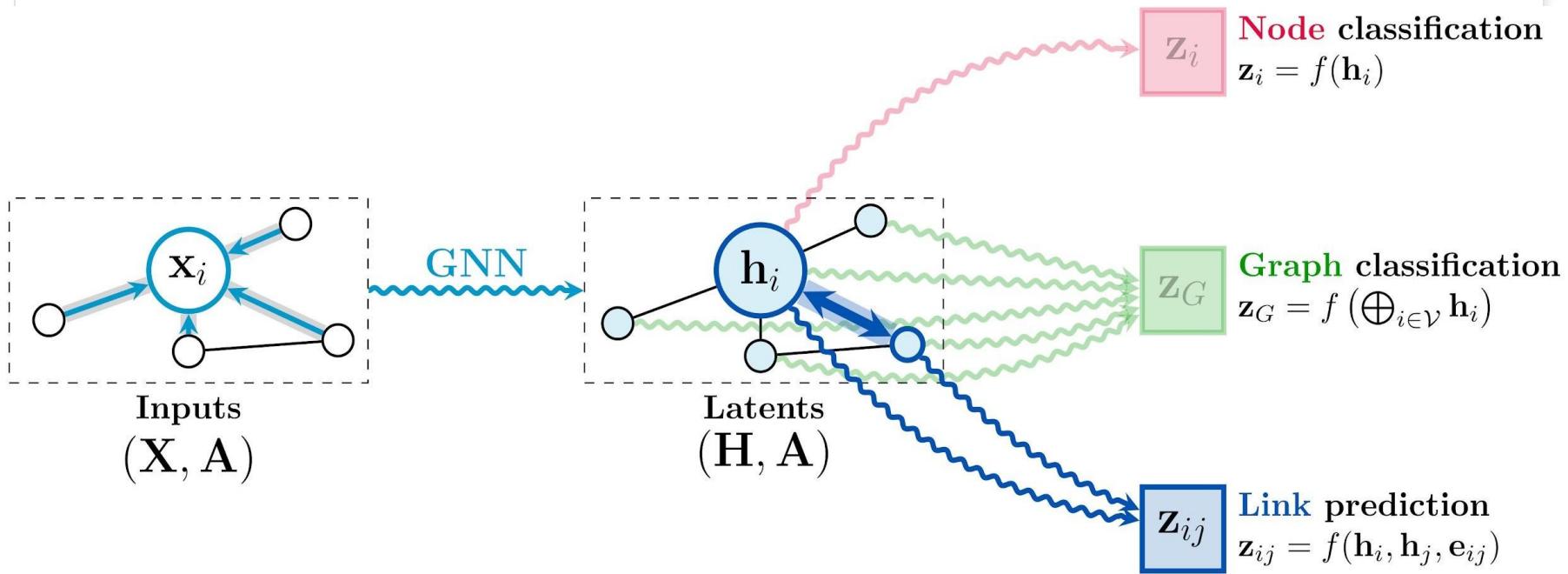
- **Graph Regression:** Predicting a continuous metric for the whole graph.

- **Example (Traffic):** In the DeepMind Google Maps study, the model performed “graph regression” to predict the “estimated time of arrival (ETA)” for a route.

- Instead of a class, the output is a continuous time value (e.g., ‘24 minutes’).

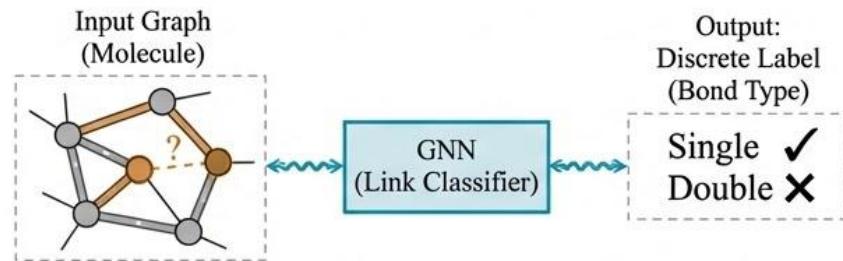


# How to use GNNs?



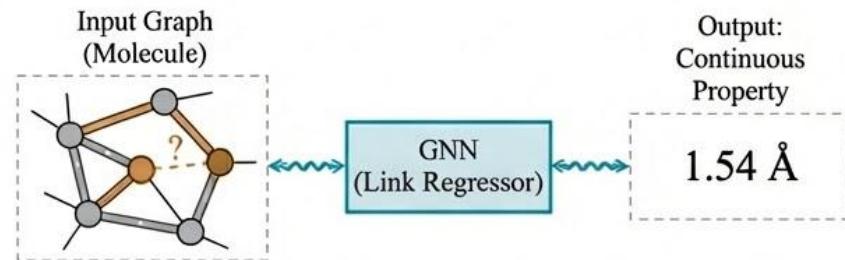
# Link Classification Example

- **Link Classification:** Predicting the type of relationship between two nodes.
- **Example (Chemistry):** Predicting the “bond type” (Single vs. Double) between two atoms if the molecular structure were only partially known.



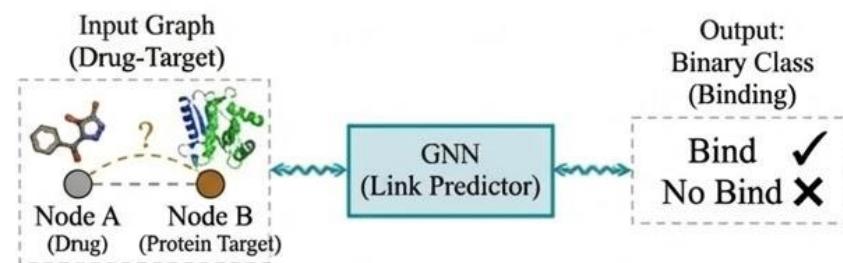
# Link Regression Example

- **Link Regression:** Predicting a continuous property of the interaction.
- **Example:** Predicting the physical bond length (distance in Angstroms) or the interaction strength energy between two atoms.



# Link Prediction (Existence) Example

- **Link Prediction (Existence):** A special case of classification used to predict missing edges.
- **Example:** In a drug-target interaction graph, predicting if a known drug molecule (Node A) will bind to a specific protein target (Node B).



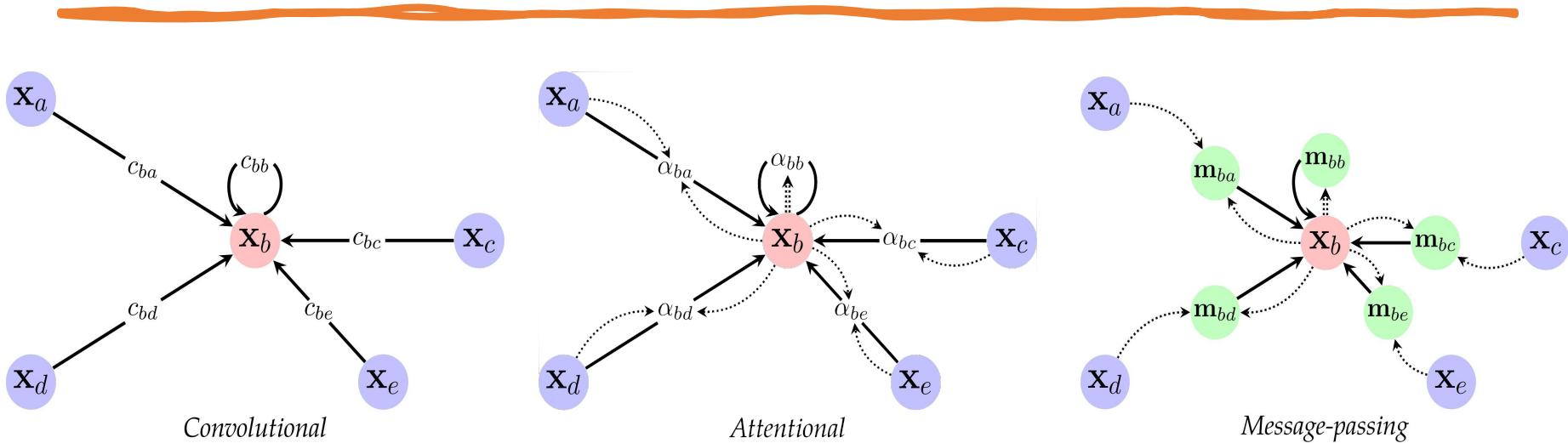
# Message Passing on Graphs

---

# What's in a GNN layer?

- We construct permutation-equivariant functions  $f(\mathbf{X}, \mathbf{A})$  over graphs by shared application of a local permutation-invariant  $g(\mathbf{x}_i, \mathbf{X}_{N_i})$ .
  - We often refer to  $f$  as “GNN layer”,  $g$  as “diffusion”, “propagation”, “message passing”
- Now we look at ways in which we can actually and concretely **define**  $g$ .
  - **Very intense** area of research!
- Fortunately, *almost all* proposed layers can be classified as one of three *spatial “flavours”*.

# The 3 “flavours” of GNN layers



$$\mathbf{h}_i = \phi \left( \mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij} \psi(\mathbf{x}_j) \right)$$

$$\mathbf{h}_i = \phi \left( \mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j) \psi(\mathbf{x}_j) \right)$$

$$\mathbf{h}_i = \phi \left( \mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j) \right)$$

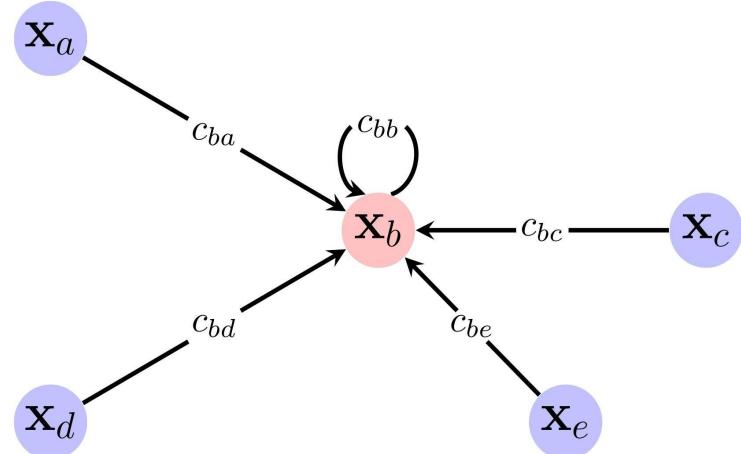


# Convolutional GNN

- Features of neighbours aggregated with fixed weights,  $c_{ij}$

$$\mathbf{h}_i = \phi \left( \mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij} \psi(\mathbf{x}_j) \right)$$

- Usually, the weights depend directly on  $\mathbf{A}$ .
  - ChebyNet (Defferrard *et al.*, NeurIPS'16)
  - GCN (Kipf & Welling, ICLR'17)
- Useful for **homophilous** graphs and **scaling up**
  - When edges encode *label similarity*

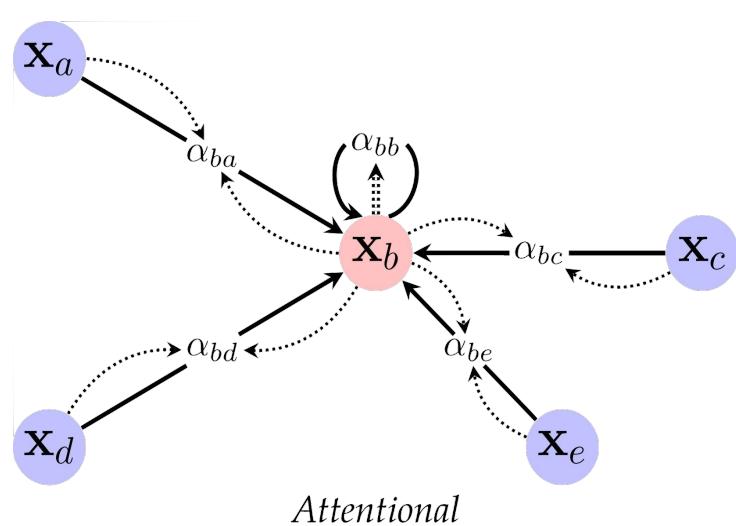


# Attentional GNN

- Features of neighbours aggregated with **implicit weights** (via *attention*)

$$\mathbf{h}_i = \phi \left( \mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j) \psi(\mathbf{x}_j) \right)$$

- Attention weight computed as  $\alpha_{ij} = a(\mathbf{x}_i, \mathbf{x}_j)$ 
  - MoNet (Monti *et al.*, CVPR'17)
  - GAT (Veličković *et al.*, ICLR'18)
- Useful as “middle ground” w.r.t. **capacity** and **scale**
  - Edges need not encode homophily
  - But still compute *scalar* value in each edge

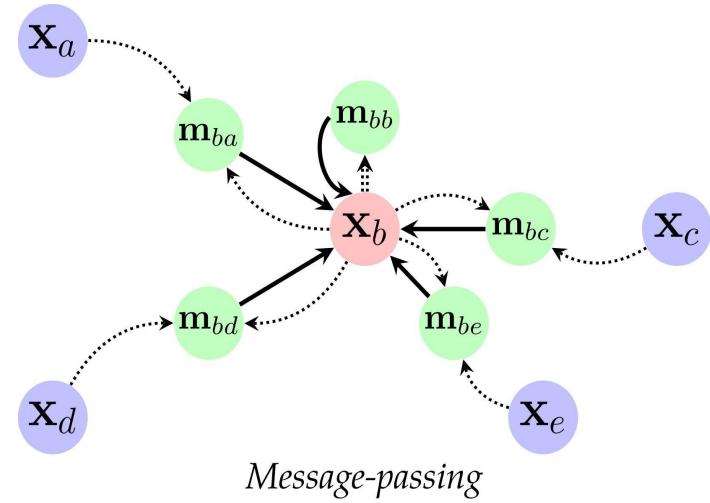


# Message-passing GNN

- Compute **arbitrary vectors** (“messages”) to be sent across edges

$$\mathbf{h}_i = \phi \left( \mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j) \right)$$

- Messages computed as  $\mathbf{m}_{ij} = \psi(\mathbf{x}_i, \mathbf{x}_j)$ 
  - Interaction Networks (Battaglia *et al.*, NeurIPS’16)
  - MPNN (Gilmer *et al.*, ICML’17)
- Most **generic** GNN layer
  - May have *scalability* or *learnability* issues
  - Ideal for *computational chemistry, reasoning and simulation*



# Rich ecosystem of libraries



PyTorch  
geometric

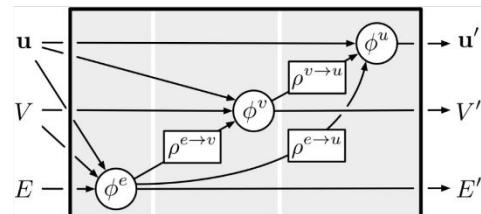
[github.com/rusty1s/pytorch\\_geometric](https://github.com/rusty1s/pytorch_geometric)

DGL  
[dgl.ai](https://dgl.ai)

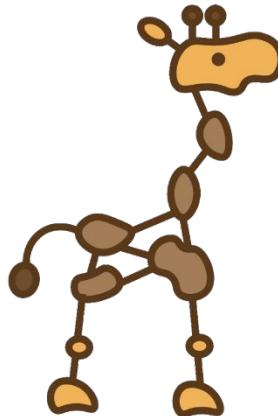


Spektral

[graphneural.network](https://graphneural.network)



[github.com/deepmind/graph\\_nets](https://github.com/deepmind/graph_nets)



[github.com/deepmind/jraph](https://github.com/deepmind/jraph)

# Rich ecosystem of datasets



[ogb.stanford.edu](http://ogb.stanford.edu)



<https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html>



TUDataset

[graphlearning.io](http://graphlearning.io)

## Benchmarking Graph Neural Networks

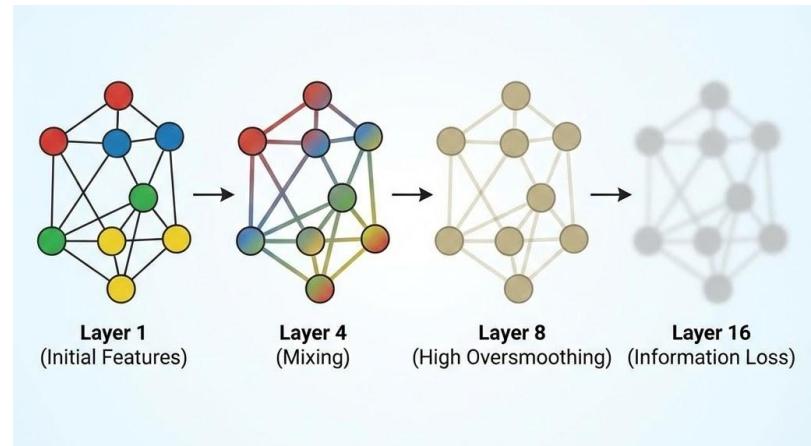
[github.com/graphdeeplearning/benchmarking-gnns](https://github.com/graphdeeplearning/benchmarking-gnns)

# GNN limitations

---

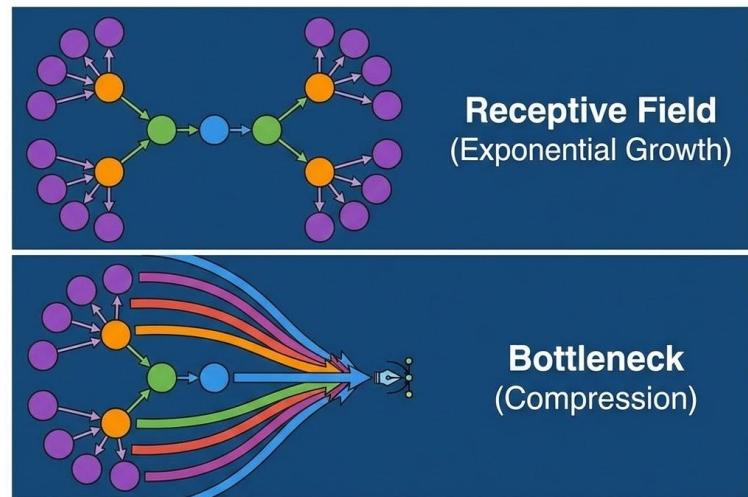
# Oversmoothing problem

- **Mechanism:** With each layer, a node mixes its features with its neighbors. As layers increase, the "receptive field" grows.
- **Consequence:** Eventually, the aggregated features for *all* nodes in a connected component begin to converge to the same average value. The nodes become indistinguishable from one another, and classification performance drastically drops.



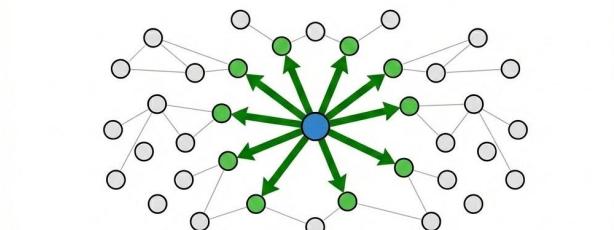
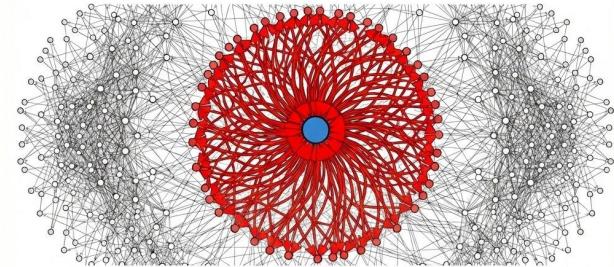
# Oversquashing problem

- **Mechanism:** In message-passing GNNs, the receptive field grows exponentially with each layer (e.g., 2 neighbors → 4 → 8...), leading to a flood of information from distant nodes.
- **Consequence:** To update a node's representation, the GNN must compress all incoming information into a single, fixed-size vector, creating a bottleneck.
- **Result:** Signals from faraway nodes become diluted or lost ("squashed"), making it hard for the model to learn long-range interactions—like between distant atoms in a large protein.



# Scaling up problem

- **Problem:** In a 3-layer GNN, computing a node's embedding requires its neighbors and their neighbors.
  - In dense graphs, this recursive expansion can quickly cover the entire dataset, causing GPU memory overflow.
- **Solution (GraphSAGE):** the model samples a fixed number (e.g., 10 or 25) at each hop
  - This keeps the computational graph size predictable and manageable.
  - Enables GNNs to scale efficiently to massive datasets, regardless of node degree.

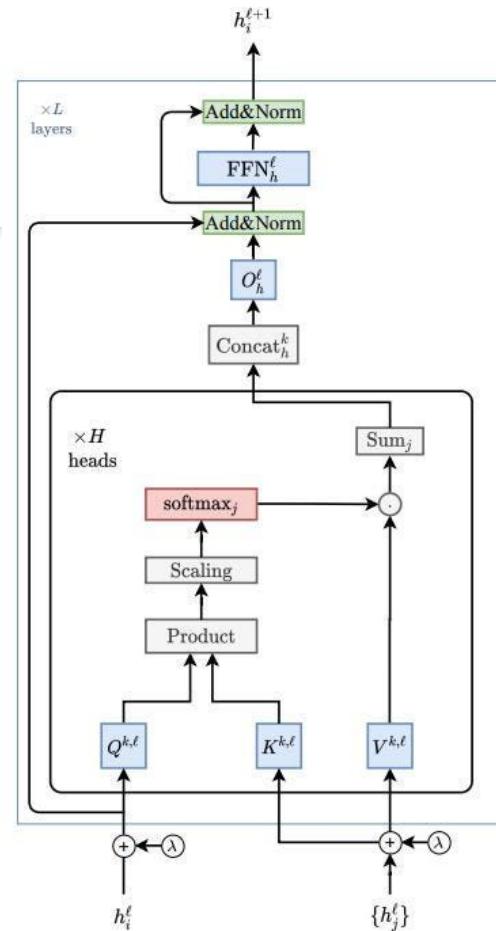


# Novel trends in GNNs

---

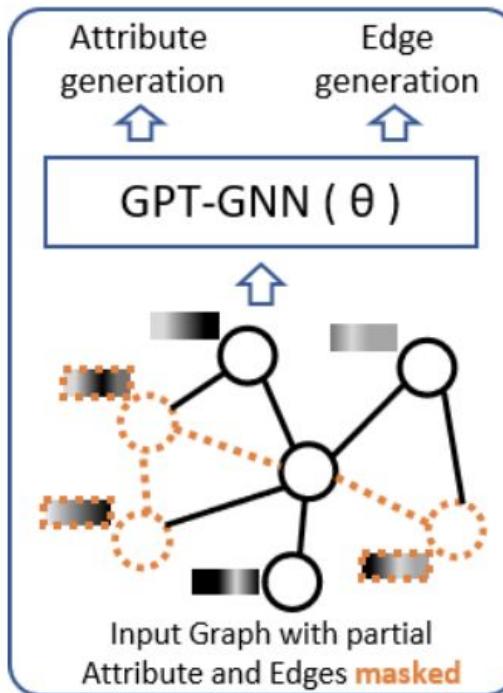
# Graph Transformer

- **Mechanism:** Graph Transformers allow nodes to attend to *any* other node in the graph, softly constrained by the adjacency matrix.
- **Positional Encodings:** Since graphs lack a linear order (unlike text), these models rely on sophisticated structural encodings (e.g., Laplacian eigenvectors) to inform the model about the graph's topology.
- **Impact:** These models have shown particular promise in molecular property prediction and reasoning tasks where distant interactions are crucial.



# GNN Generation

- **Applications:** In material science and drug discovery, these models can generate stable crystal structures or novel molecular candidates by learning the rules of valid chemical bonds.
- **Techniques:** Approaches include diffusion models adapted for graphs and autoregressive models that build graphs node-by-node.



# Thanks for listening!

## Any questions?

---