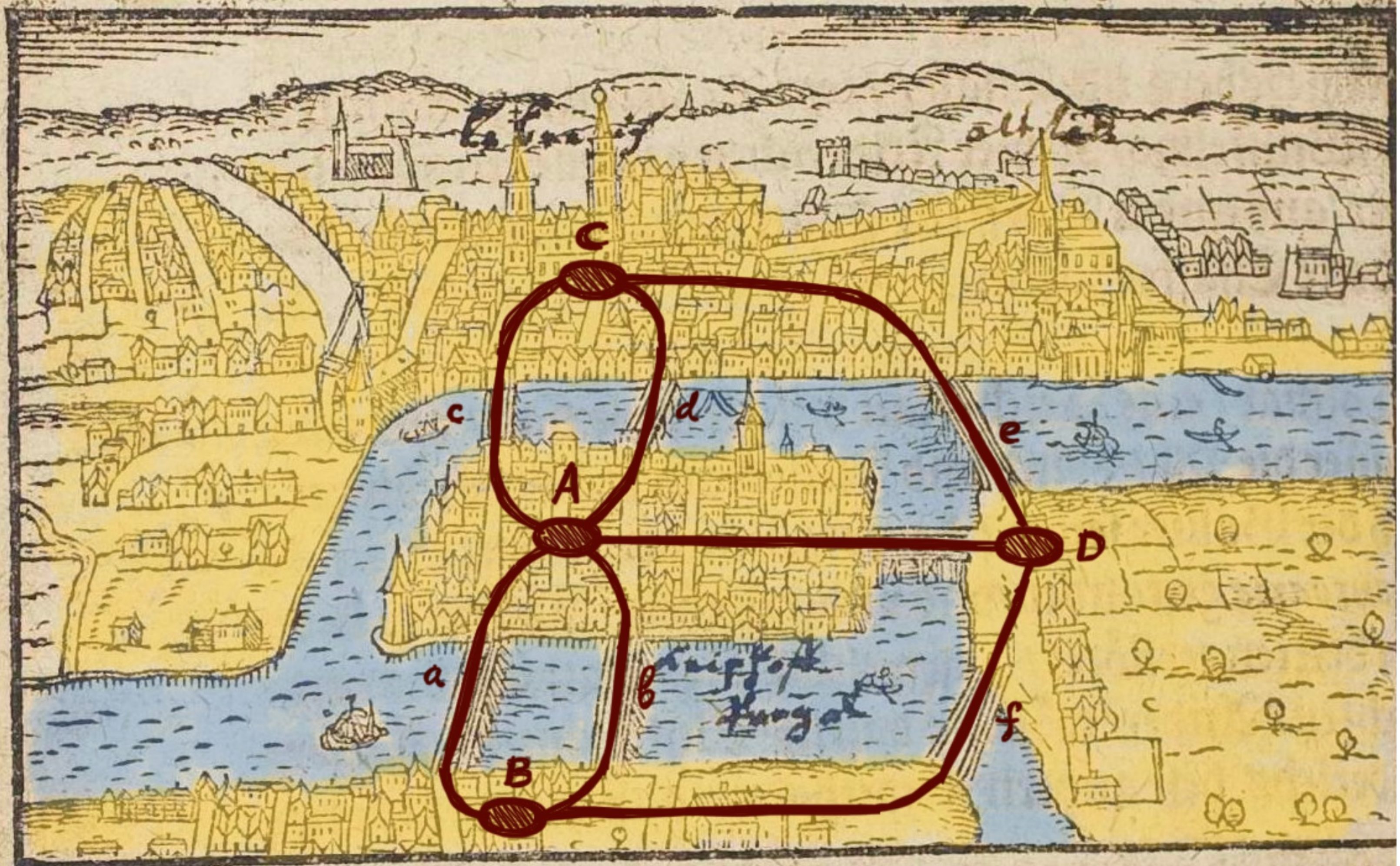
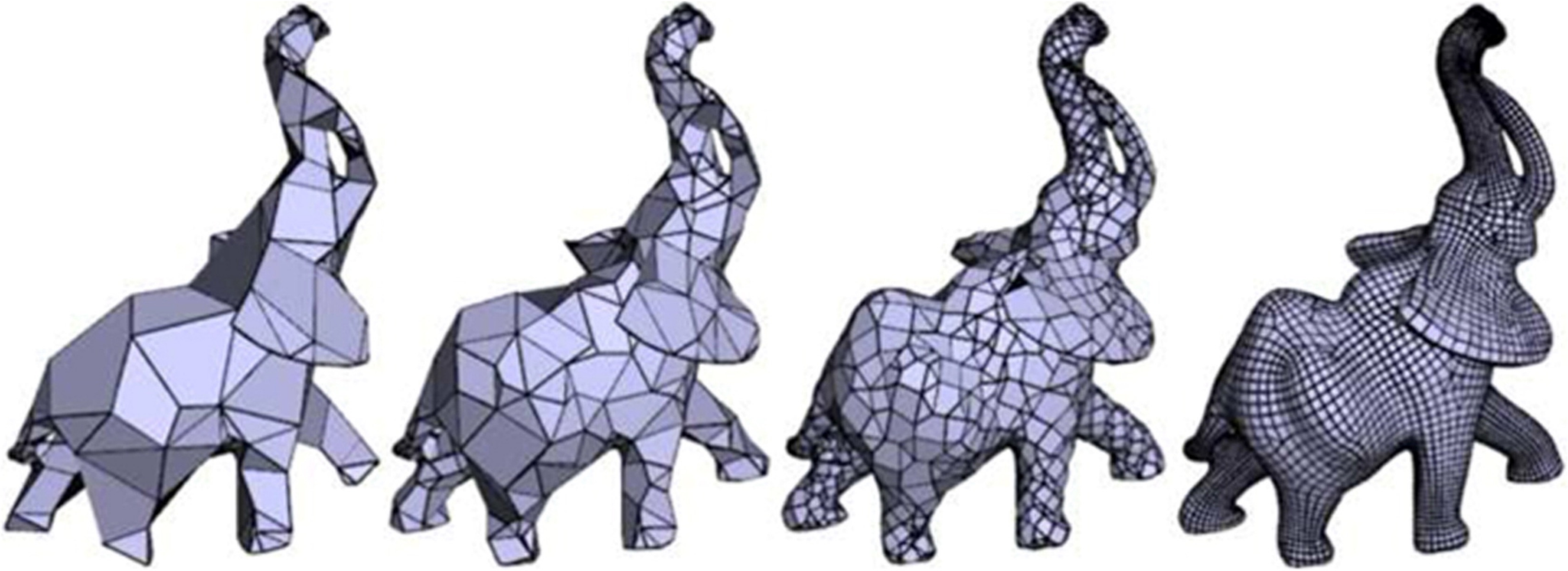
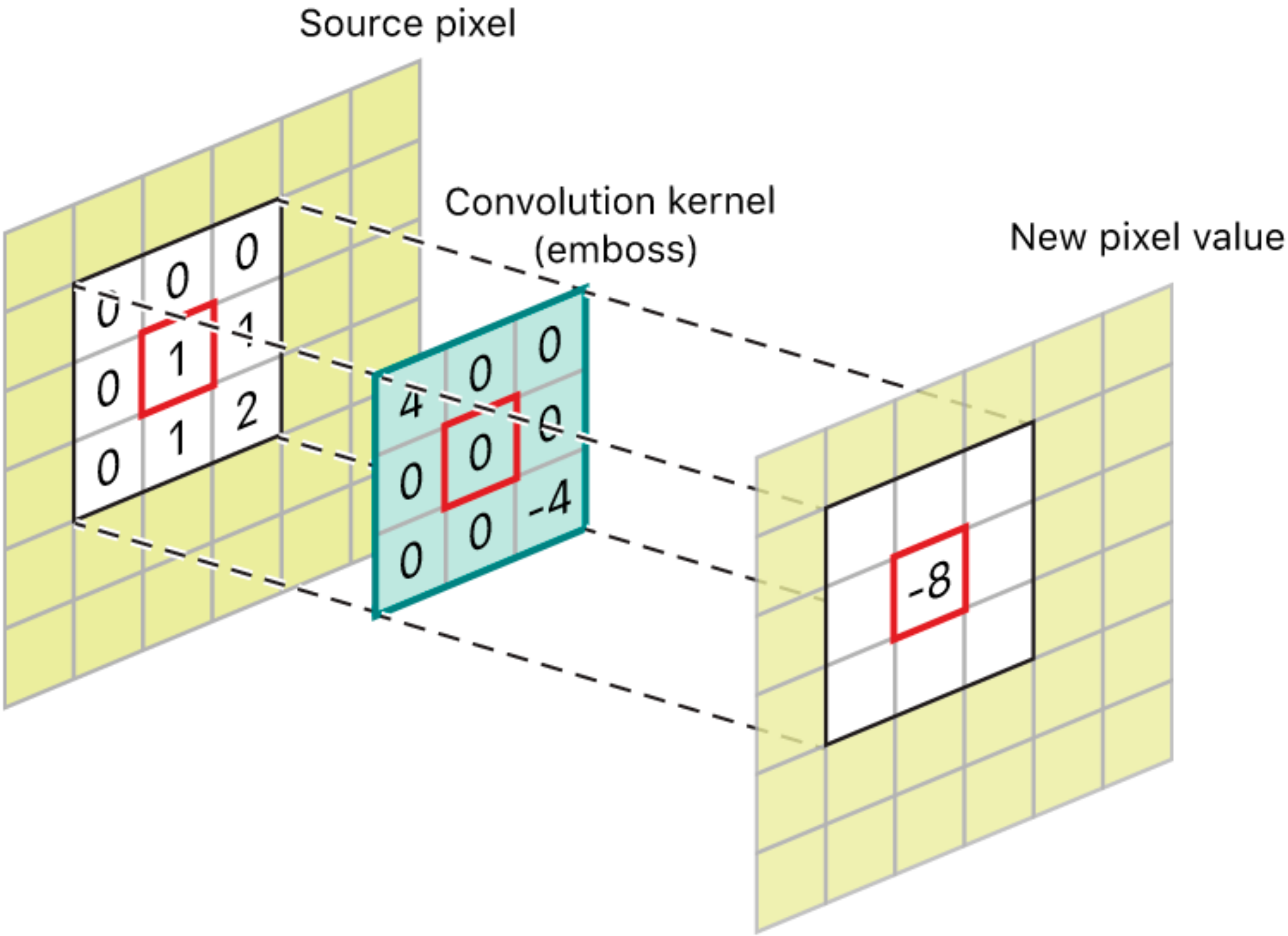
# A Continuous Perspective on Graph Neural Networks

**Ben Chamberlain**

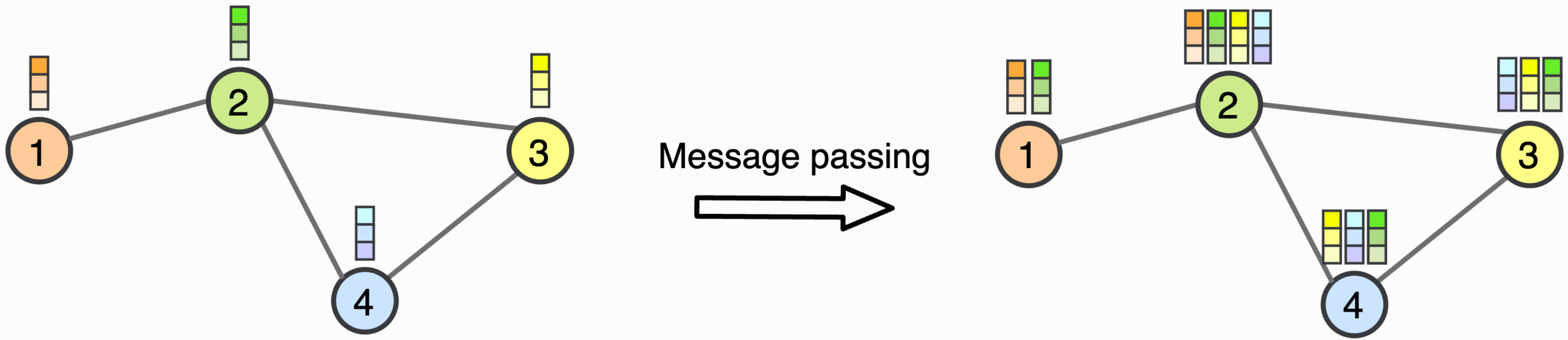
@b_p_chamberlain


C
c
d
e
A
D
a
b
f
B




Source pixel
Convolution kernel
(emboss)
New pixel value
0
0
0
1
1
0
2
1
0
0
0
0
4
0
0
0
-4
0
0
0
-8

# Message Passing GNNs



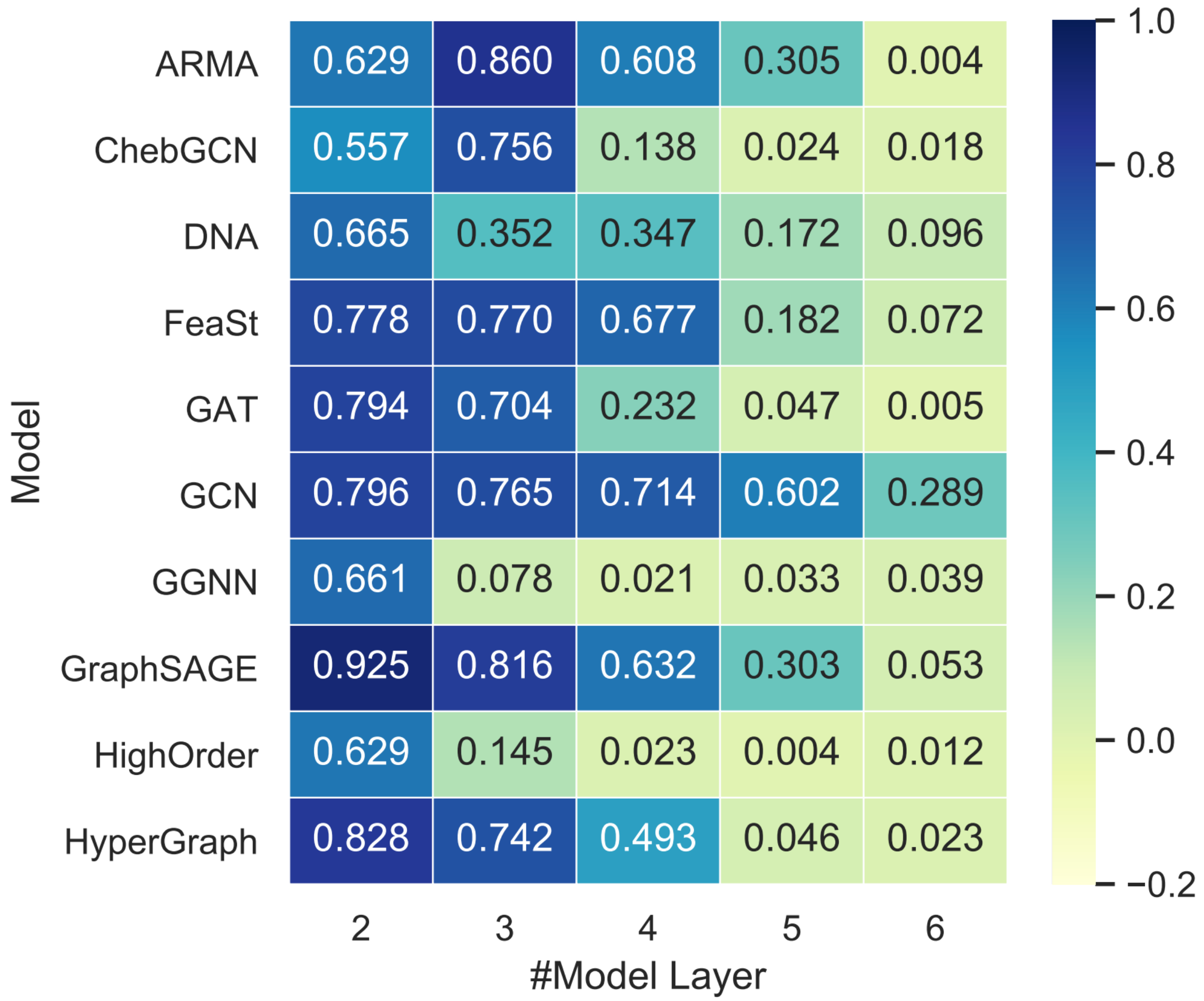

- **Most GNNs are Message Passing GNNs**
- **Message Passing GNNs interleave two steps**
  1. **Propagation along edge (message passing)**
  2. **Shared feature transformation by a neural network**

$$\mathbf{H}^{(k+1)} = \sigma\left(\underbrace{\mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}}_{\text{Message passing}}\underbrace{\mathbf{H}^{(k)}\mathbf{W}_k}_{\text{Feature transformation}}\right)$$

# *What's Wrong with (Message Passing) GNNs?*

- **Deep GNNs ⇒ over-smoothing**
- **Input graph = computational graph ⇒ bottlenecks & "over-squashing"**
- Graph & features incompatible structure ("heterophily") ⇒ poor performance
- Limited expressive power ⇒ cannot detect simple structures such as cycles or cliques

# Oversmoothing

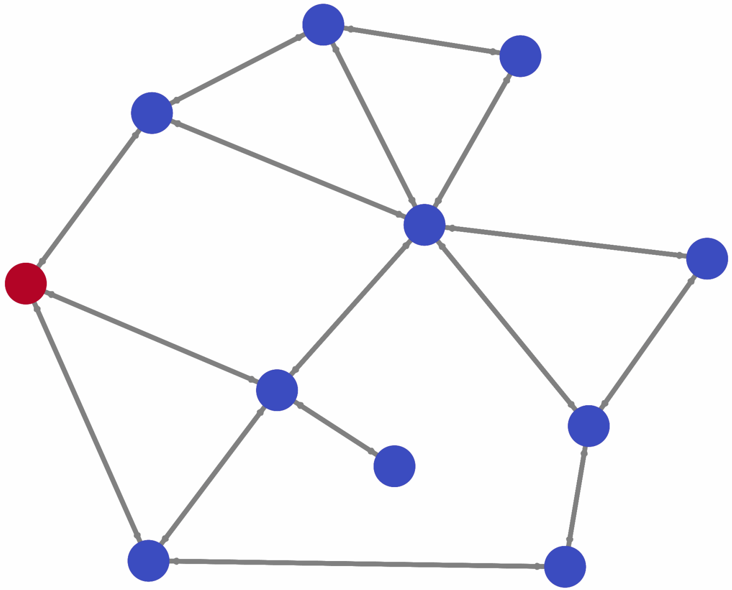

ARMA 0.629 0.860 0.608 0.305 0.004
ChebGCN 0.557 0.756 0.138 0.024 0.018
DNA 0.665 0.352 0.347 0.172 0.096
FeaSt 0.778 0.770 0.677 0.182 0.072
GAT 0.794 0.704 0.232 0.047 0.005
GCN 0.796 0.765 0.714 0.602 0.289
GGNN 0.661 0.078 0.021 0.033 0.039
GraphSAGE 0.925 0.816 0.632 0.303 0.053
HighOrder 0.629 0.145 0.023 0.004 0.012
HyperGraph 0.828 0.742 0.493 0.046 0.023
Model
2
3
4
5
6
#Model Layer
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Chen 2020

# Message Passing is Information Diffusion

**Laplacian diffusion on graphs:**

$$X_t = LX_{t-1}$$

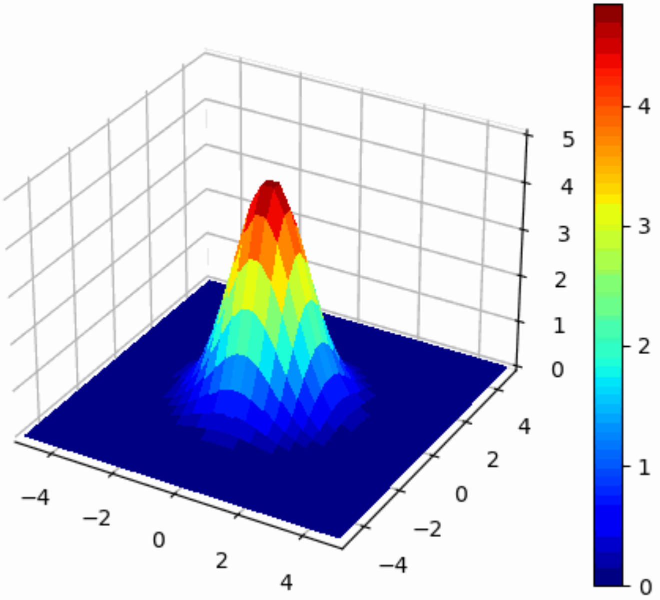


$$\mathbf{E}(\mathbf{X}) = \frac{1}{v} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \|\mathbf{X}_i - \mathbf{X}_j\|^2$$

Pierre-Simon Laplace

# The continuous analogy is heat diffusion

**Heat diffusion:**




Joseph Fourier

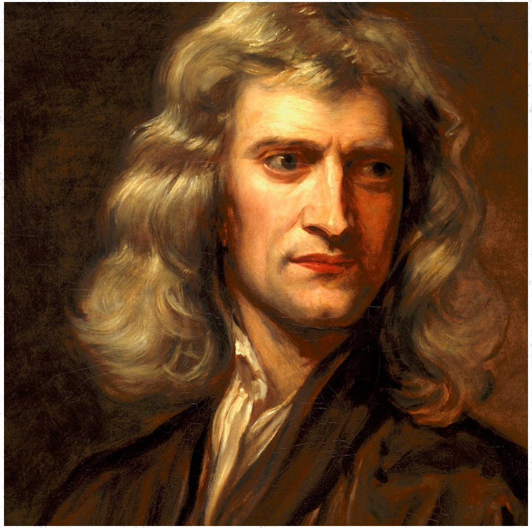
$$\frac{\partial x}{\partial t} = \Delta x$$

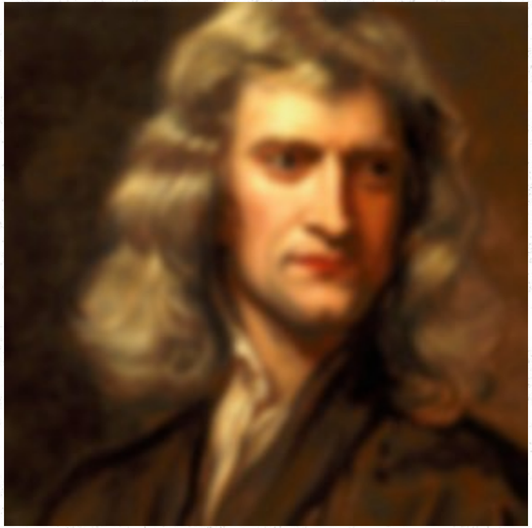
$$\mathcal{E}(f,h) = \frac{1}{2}\int_{\mathbb{R}^n} \|\nabla f\|_h^2 \, dx$$

# Diffusion in Image Processing

$$\frac{\partial}{\partial t}x = c\Delta x$$

$$\frac{\partial}{\partial t}x = \text{div}\left[\frac{1}{1+(|\nabla x|/\lambda)^2}\nabla \boldsymbol{x}\right]$$





Homogeneous diffusion

Non-homogeneous diffusion

Perona, Malik 1990; Sochen et al. 1998; Tomasi, Manduchi 1998; Weickert 1998; Buades et al. 2005

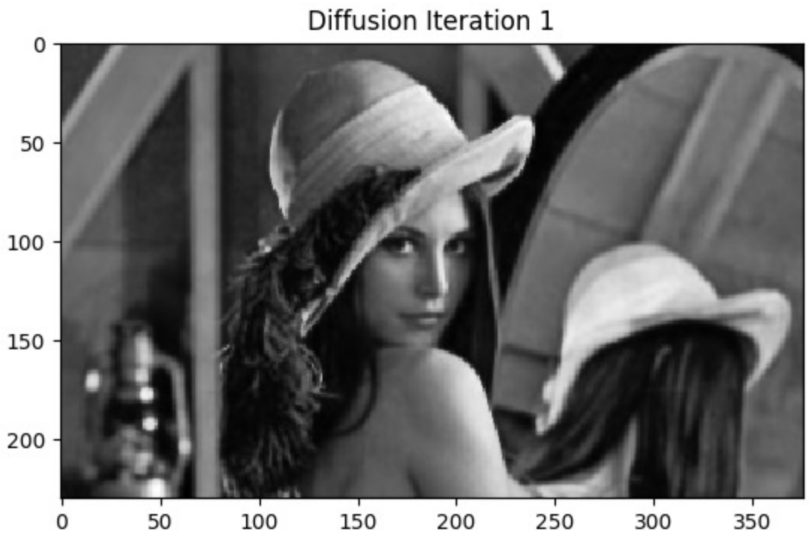
# Variational methods in image processing – 50 frame diffusion

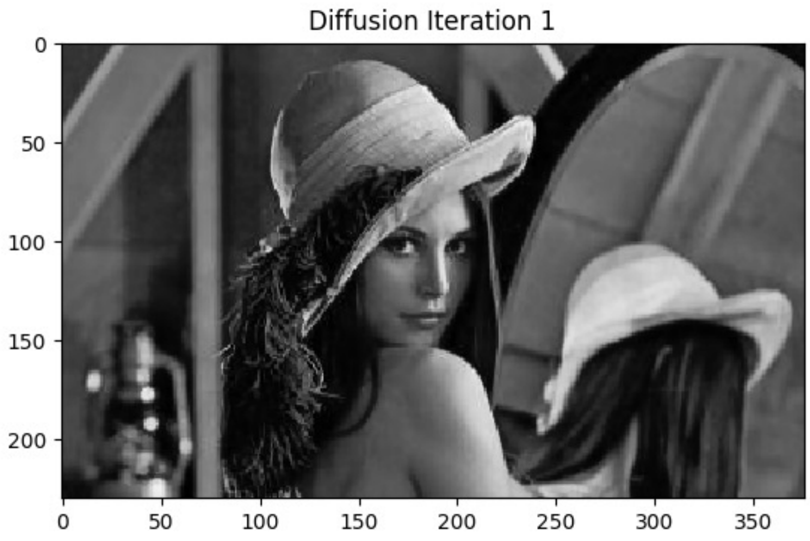
**Gaussian:**

$$g(x) = c$$

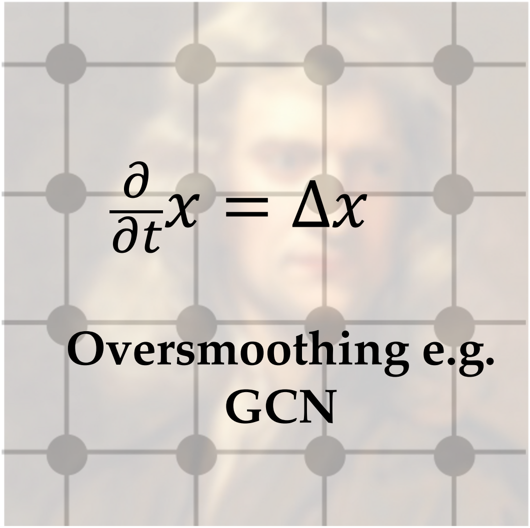
**Perona & Malik:**

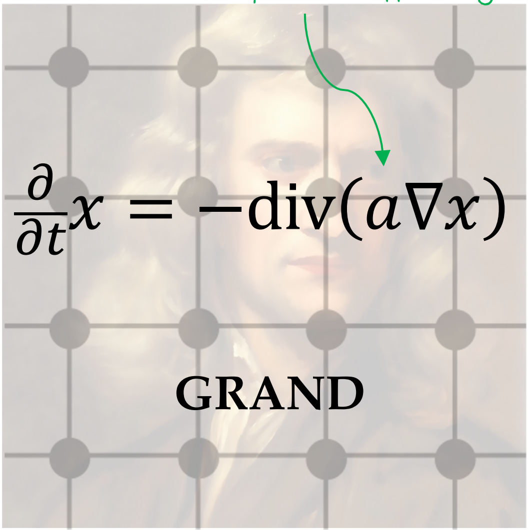
$$g(|\nabla x|) = \frac{1}{1 + (|\nabla x|/\lambda)^2}$$

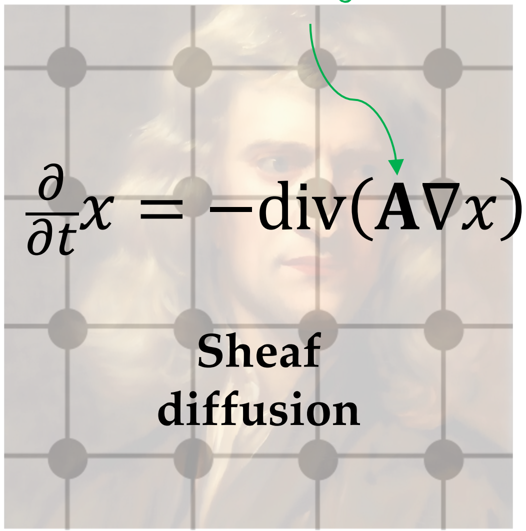


# Attentional diffusivity for GNNs




Kipf 2017, Chamberlain 2021a, Bodner 2022

For more on our work on sheaves see: Aleksa Gordic's YouTube channel The AI Epiphany https://www.youtube.com/c/TheAIEpiphany

## Spatial discretisation: Continuous to Discrete

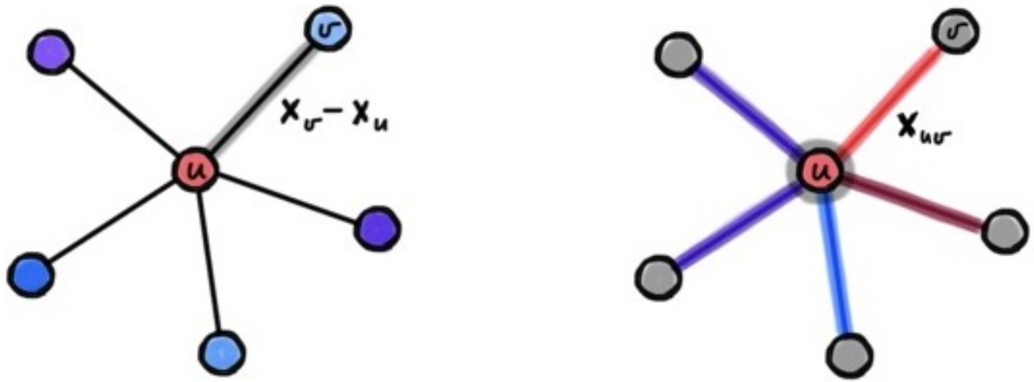
$$\frac{\partial \boldsymbol{x}(u,t)}{\partial t} = \mathrm{div}[\nabla \boldsymbol{x}(u,t)] \qquad (1)$$

**gradient - flow along edges**

$$(\nabla \boldsymbol{X})_{uv} = \boldsymbol{x}_u - \boldsymbol{x}_v$$

**divergence - aggregation of edges**

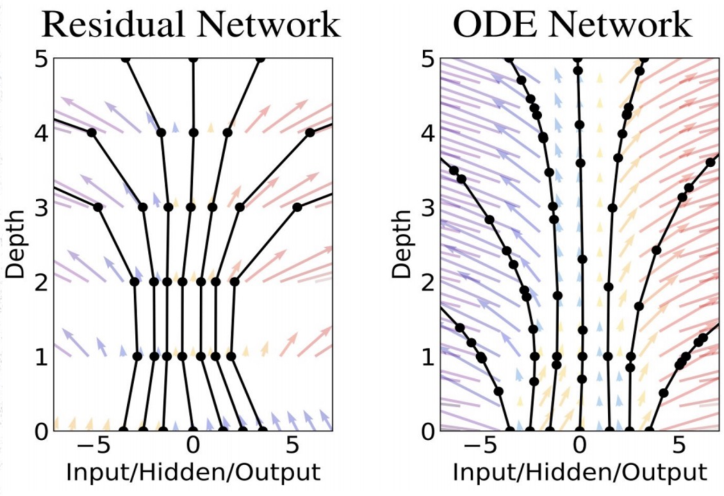
$$\big(\mathrm{div}(\boldsymbol{X})\big)_u = \sum_v w_{uv} x_{uv}$$




$$\dot{\boldsymbol{X}}(t) = \big(\boldsymbol{A}\big(\boldsymbol{X}(t)\big) - \boldsymbol{I}\big)\boldsymbol{X}(t) \quad (2)$$

# Temporal discretisation

**Neural ODEs:**




**ResNet:** $\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t)$

**Dynamics:** $\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}_t, \theta_t, t)$

**ODE Solvers:**

$$\boldsymbol{z}(t_1) = \boldsymbol{z}(t_0) + \int_{t_0}^{T} f(\boldsymbol{z}(t), t, \theta)\, dt = \text{ODESolve}\,(\boldsymbol{z}(t_0), f, t_0, t_1, \theta)$$

Chen et al. 2019; Haber et al. 2019, Xhonneux, 2020.

## Simple GNN by discretising time in the graph diffusion equation:

$$\dot{\boldsymbol{X}}(t) = \left(\boldsymbol{A}\big(\boldsymbol{X}(t)\big) - \boldsymbol{I}\right) \boldsymbol{X}(t) \quad (2)$$

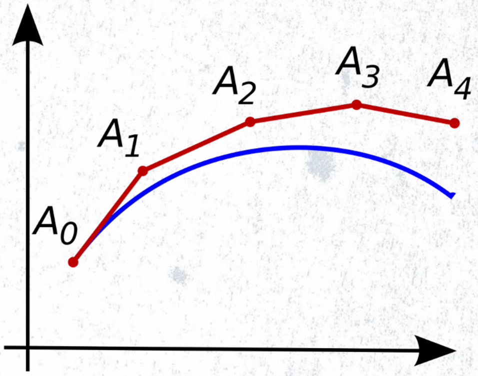
**Explicit Euler temporal discretisation**

$$\boldsymbol{X}(k+1) = \boldsymbol{X}(k) + \tau\left[\boldsymbol{A}\big(\boldsymbol{X}(k)\big) - \boldsymbol{I}\right]\boldsymbol{X}(k)$$

Set time step $\tau = 1$ get simplified GCN

$$\boldsymbol{X}(k+1) = \boldsymbol{A}\big(\boldsymbol{X}(k)\big)\boldsymbol{X}(k)$$

without feature channel mixing and non-linearities




Kipf 2017

# Better ODE Solvers

## Multi-step methods:

$$x_{n+1} + \sum_{i=1}^{s} \alpha_i x_{n+1-i} = \tau \sum_{i=0}^{s} \beta_i f_{n+1-i}$$

## Adaptive step-size solvers:

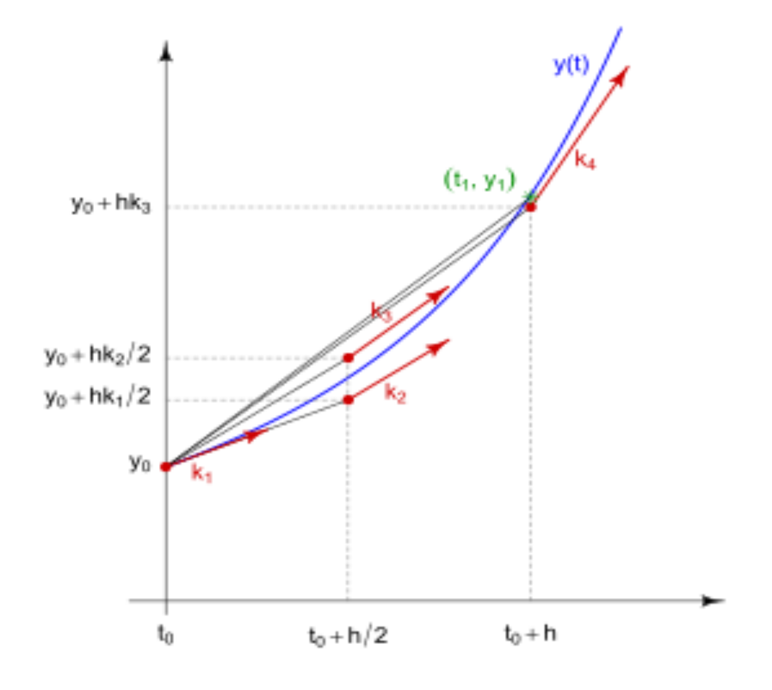
order p: $x_p(\tau)$

order p-1: $x_{p-1}(\tau)$

error: $\varepsilon = x_p(\tau) - x_{p-1}(\tau)$

tolerance: $etol = atol + rtol * \max(|x_0|, |x_1|)$

Eg: Runge-Kutta 4




If $\varepsilon > etol$
then decrease step-size $\tau$

# GRAND depth analysis:

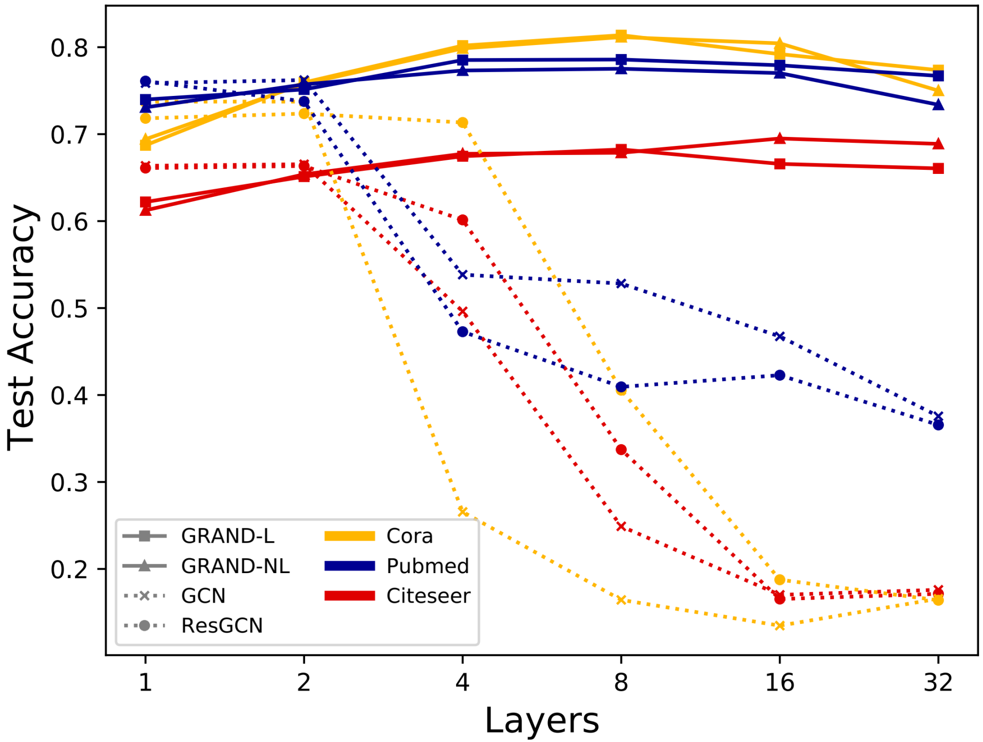


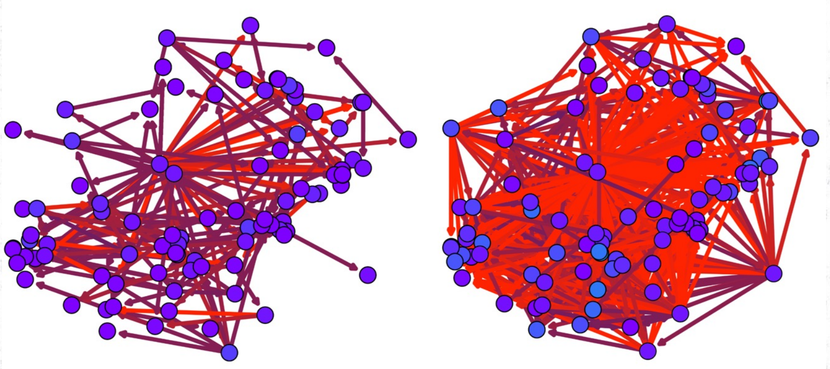

*Figure 2.* Performance of architectures of different depth.

# Decoupling the Input graph and the computational graph

# Spatial discretisation: Graph Rewiring

**Decouple input graph from information propagation graph**

- Neighbourhood sampling (GraphSAGE)[1]
- Multi-hop filters (SIGN)[2]
- Complete graph[3]
- Topology diffusion (DIGL)[4]



[1]Hamilton et al. 2017; [2]Rossi, Frasca, et B. 2020; [3]Alon, Yahav 2020; [4]Klicpera et al. 2019

# Implicit Solvers

## Implicit Versus Explicit Euler's Methods

- **Explicit Euler's Method:**

$$\frac{dy}{dx} = f(x, y)$$

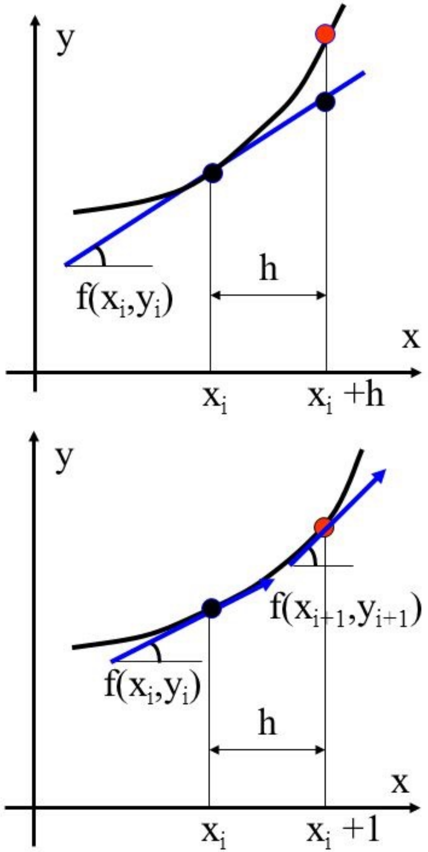
$$y_{i+1} = y_i + f(x_i, y_i) \cdot h$$

- **Implicit Euler's Method:**

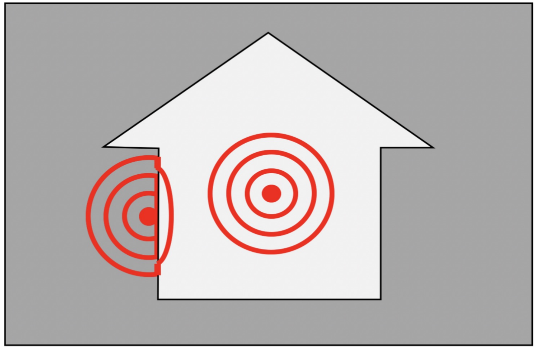
$$\frac{dy}{dx} = f(x, y)$$

$$y_{i+1} = y_i + f(x_{i+1}, y_{i+1}) \cdot h$$



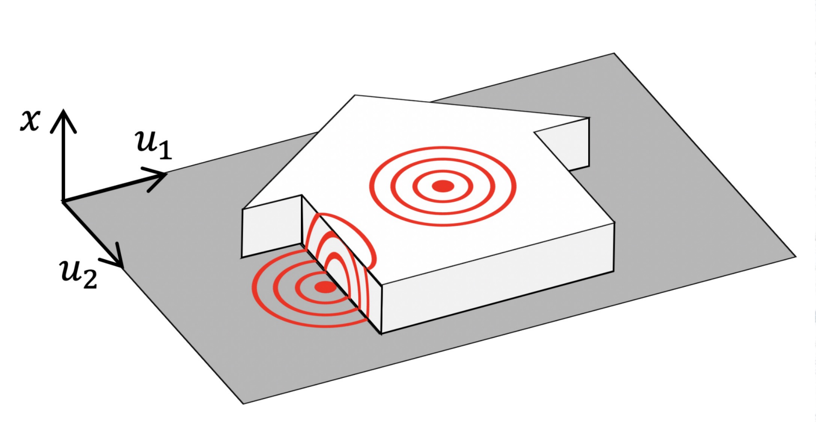


# *Images as embedded manifolds*



$$\frac{\partial}{\partial t}\mathbf{x} = -\mathrm{div}(a(\mathbf{x})\nabla\mathbf{x})$$

Non-linear diffusion



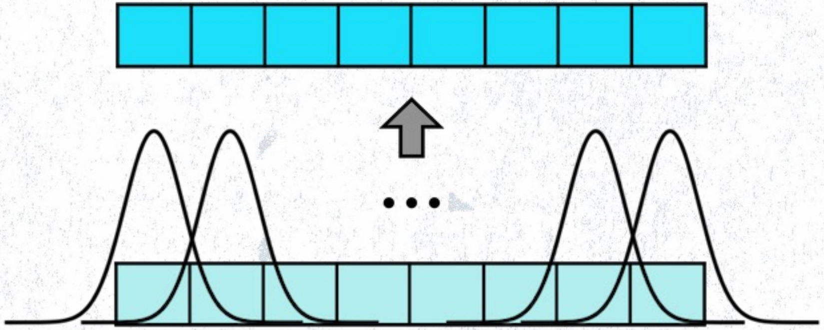

$$\frac{\partial}{\partial t}\mathbf{z} = \Delta_{\mathbf{G}}\mathbf{z}$$

Non-Euclidean diffusion

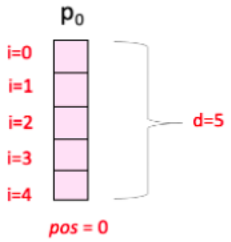
Sochen et al. 1998

# *Intro to positional encodings*

- Popularized in the transformer
- Now widely used in GNNs
- Many options
  - Random node features[1]
  - Graph Laplacian eigenvectors[2]
  - Graph substructure counts[3]
  - Bags of subgraphs[4]



$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$




[1]Sato et al. 2020; [2]Vaswani et al. 2017; Qiu et al. 2020; Dwivedi et al. 2020; [3]Bouritsas, Frasca, et B. 2020; [4]Bevilacqua, Frasca, Lim, et B., Maron 2021

# *What to do with the positional encodings*

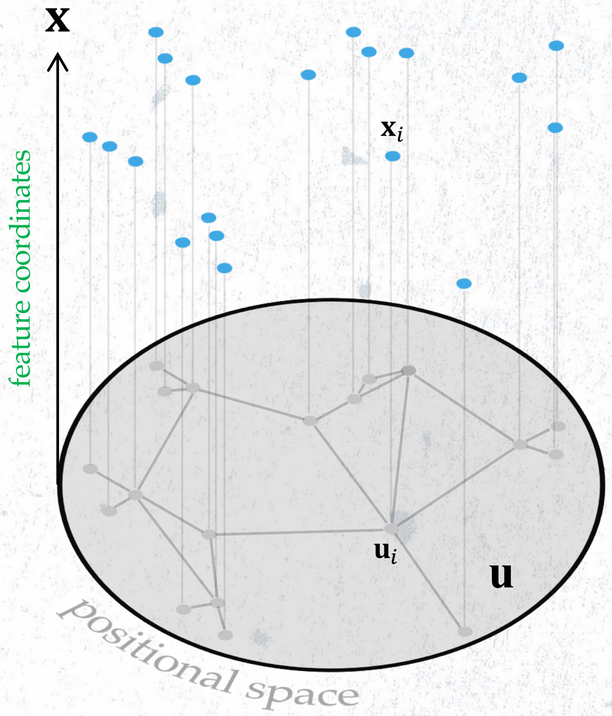
- In computer vision they are just for edge detection and then thrown away
- The graph approach is more elegant as the evolved positional encodings can be used to rewire the graph?
- Why rewire the graph?
  - Decouples the given and computational graph can remove bottlenecks or increase performance and scalability

[1]Hamilton et al. 2017; [2]Rossi, Frasca, et B. 2020; [3]Alon, Yahav 2020; [4]Klicpera et al. 2019; [5]Wang et B 2018; Kazi, Cosmo, et B. 2020

# *Graph Beltrami flow*

- Graph with positional and feature node coordinates $\mathbf{z}_i = (\mathbf{u}_i, \mathbf{x}_i)$
- **Graph Beltrami flow**

$$\frac{\partial}{\partial t}\mathbf{z}_i = \sum_{j:(i,j)\in E} a(\mathbf{z}_i, \mathbf{z}_j)(\mathbf{z}_j - \mathbf{z}_i)$$



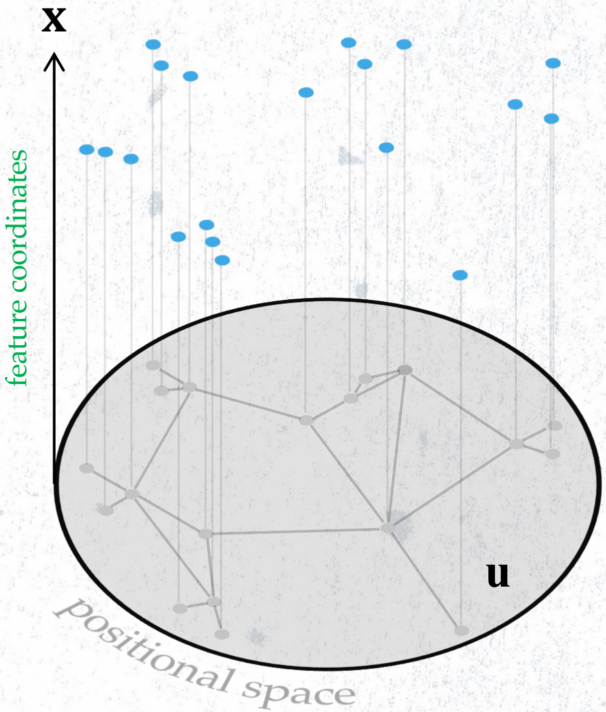

Chamberlain et B. 2021

# *Graph Beltrami flow*

- Graph with positional and feature node coordinates $\mathbf{z}_i = (\mathbf{u}_i, \mathbf{x}_i)$
- **Graph Beltrami flow**

$$\frac{\partial}{\partial t}\mathbf{z}_i = \sum_{j:(i,j)\in E} a(\mathbf{z}_i, \mathbf{z}_j)(\mathbf{z}_j - \mathbf{z}_i)$$

  - Evolution of $\mathbf{x}$ = feature diffusion



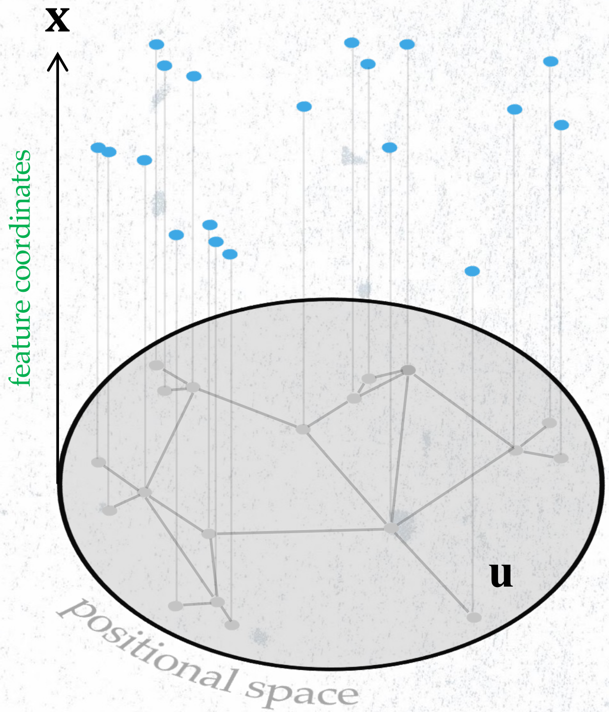

Chamberlain b. 2021

# *Graph Beltrami flow*

- Graph with positional and feature node coordinates $\mathbf{z}_i = (\mathbf{u}_i, \mathbf{x}_i)$
- **Graph Beltrami flow**

$$\frac{\partial}{\partial t}\mathbf{z}_i = \sum_{j:(i,j)\in E} a(\mathbf{z}_i, \mathbf{z}_j)(\mathbf{z}_j - \mathbf{z}_i)$$

  - Evolution of $\mathbf{x}$ = feature diffusion



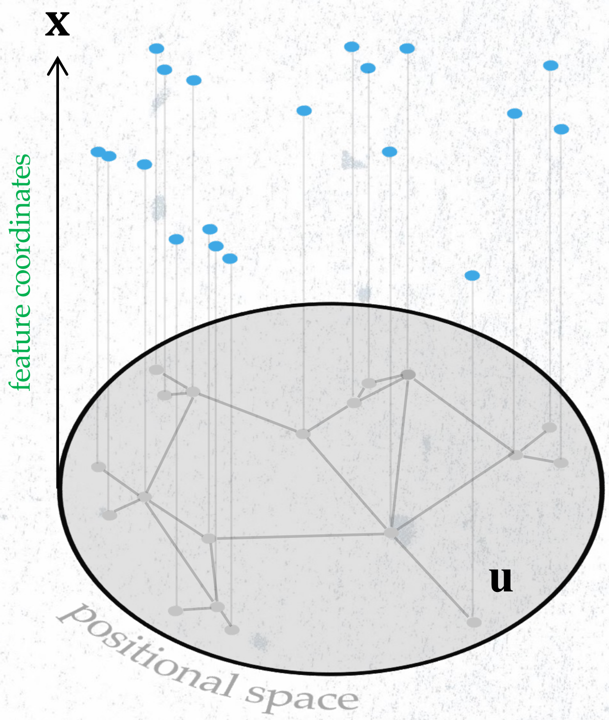

Chamberlain et B. 2021

# *Graph Beltrami flow*

- Graph with positional and feature node coordinates $\mathbf{z}_i = (\mathbf{u}_i, \mathbf{x}_i)$
- **Graph Beltrami flow**

$$\frac{\partial}{\partial t}\mathbf{z}_i = \sum_{j:(i,j)\in E} a(\mathbf{z}_i, \mathbf{z}_j)(\mathbf{z}_j - \mathbf{z}_i)$$

  - Evolution of $\mathbf{x}$ = feature diffusion
  - Evolution of $\mathbf{z}$ = graph rewiring




Chamberlain et B. 2021

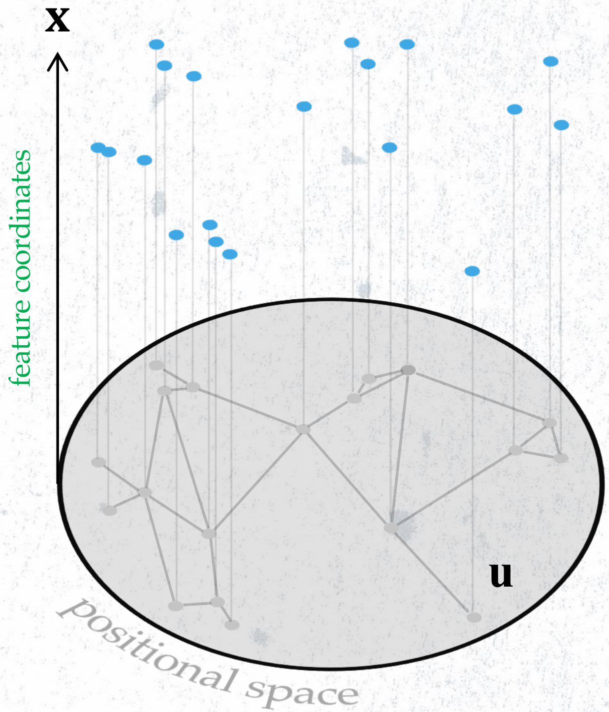
# *Graph Beltrami flow*

- Graph with positional and feature node coordinates $\mathbf{z}_i = (\mathbf{u}_i, \mathbf{x}_i)$
- **Graph Beltrami flow**

$$\frac{\partial}{\partial t}\mathbf{z}_i = \sum_{j:(i,j)\in E'} a(\mathbf{z}_i, \mathbf{z}_j)(\mathbf{z}_j - \mathbf{z}_i)$$

rewired graph

  - Evolution of $\mathbf{x}$ = feature diffusion
  - Evolution of $\mathbf{z}$ = graph rewiring



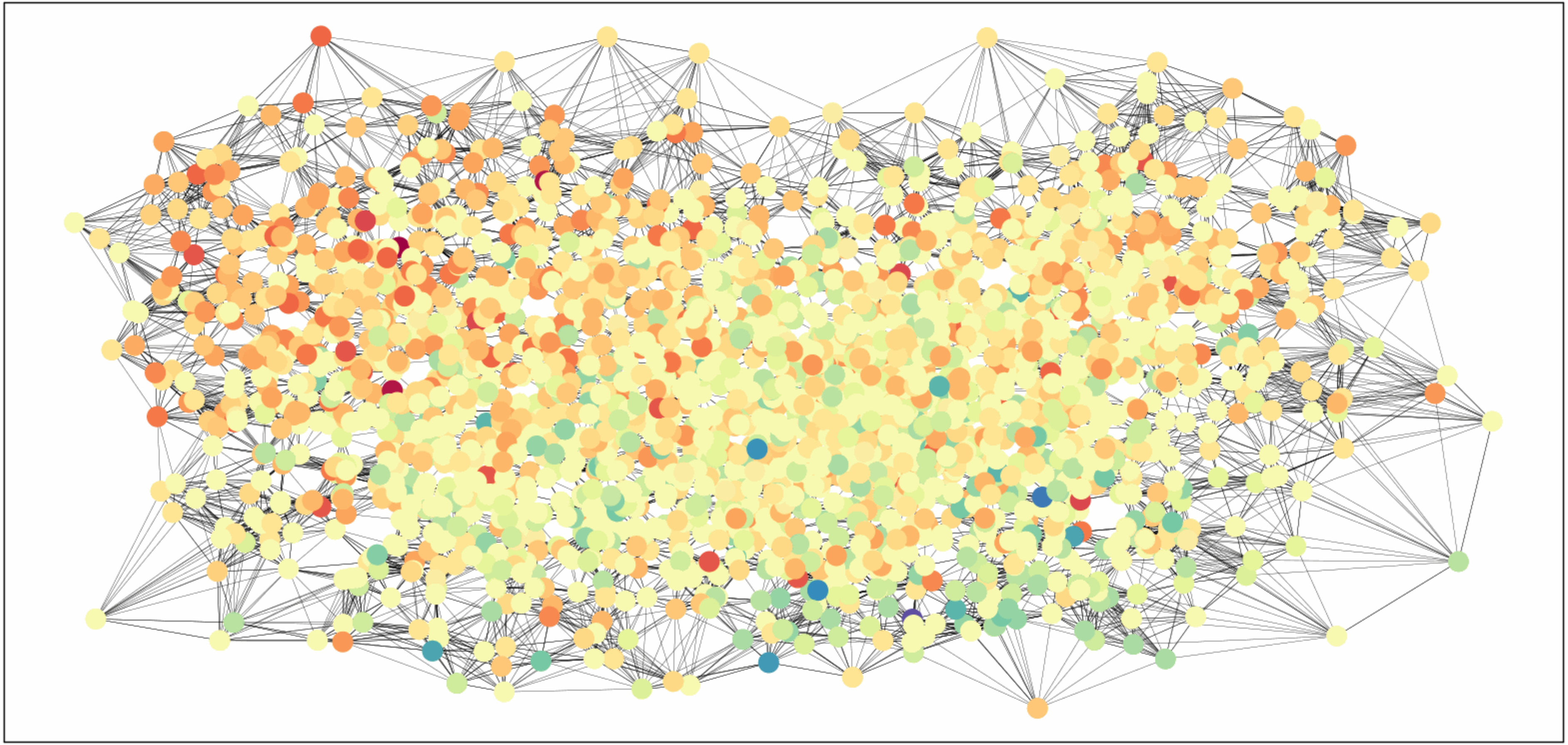

Chamberlain et al. NeurIPS 2021

# *GNN Architectures as instances of BLEND*

$$\mathbf{Z}^{(k+1)} = \mathbf{Q}\left(\mathbf{Z}^{(k)}\right)\mathbf{Z}^{(k)}$$

| Method | Evolution | Diffusivity | Graph | Discretisation |
|---|---|---|---|---|
| ChebNet | $\mathbf{X}$ | Fixed $\mathbf{A}$ | Fixed $E$ | Explicit Fixed step |
| GAT | $\mathbf{X}$ | $\mathbf{A}(\mathbf{X})$ | Fixed $E$ | Explicit Fixed step |
| MoNet | $\mathbf{X}$ | $\mathbf{A}(\mathbf{U})$ | Fixed $E$ | Explicit Fixed step |
| Transformer | $\mathbf{X}$ | $\mathbf{A}(\mathbf{U},\mathbf{X})$ | Fixed $E = V^2$ | Explicit Fixed step |
| DIGL | $\mathbf{X}$ | $\mathbf{A}(\mathbf{X})$ | Fixed $E(\mathbf{U})$ | Explicit Fixed step |
| DGCNN | $\mathbf{X}$ | $\mathbf{A}(\mathbf{X})$ | Adaptive $E(\mathbf{X})$ | Explicit Fixed step |
| BLEND | $\mathbf{U},\mathbf{X}$ | $\mathbf{A}(\mathbf{U},\mathbf{X})$ | Adaptive $E(\mathbf{U})$ | Explicit Adaptive step / Implicit |

Beltrami Flow, diffusion time=0



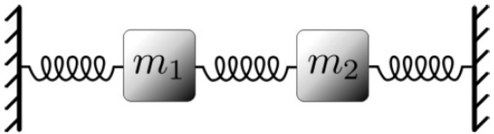
# GNNs as Interacting Particle Systems

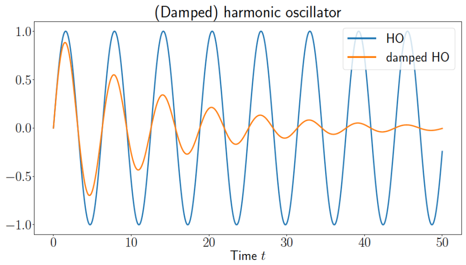
# Interacting Particle Approach

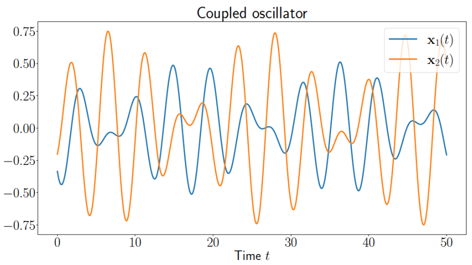
- Associate positions with node embeddings
- Learning generates node trajectories
- Trajectories are constrained by an energy and described by a differential equation (the gradient flow of the energy)

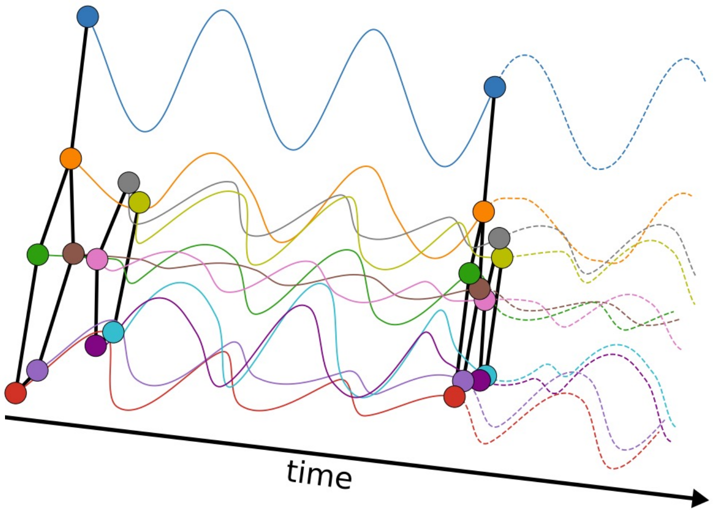
# Graph Coupled Oscillator Networks (GraphCON) 

Rusch et at ICML 22




$$\mathbf{X}'' = \sigma(\mathbf{F}_\theta(\mathbf{X}, t) + \mathbf{X}\overline{\mathbf{W}} + \overline{\mathbf{b}}) - \gamma \mathbf{X} - \alpha \mathbf{X}'$$

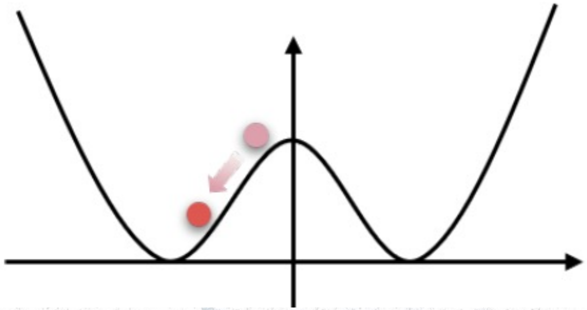
Graph coupling

Damped Oscillator

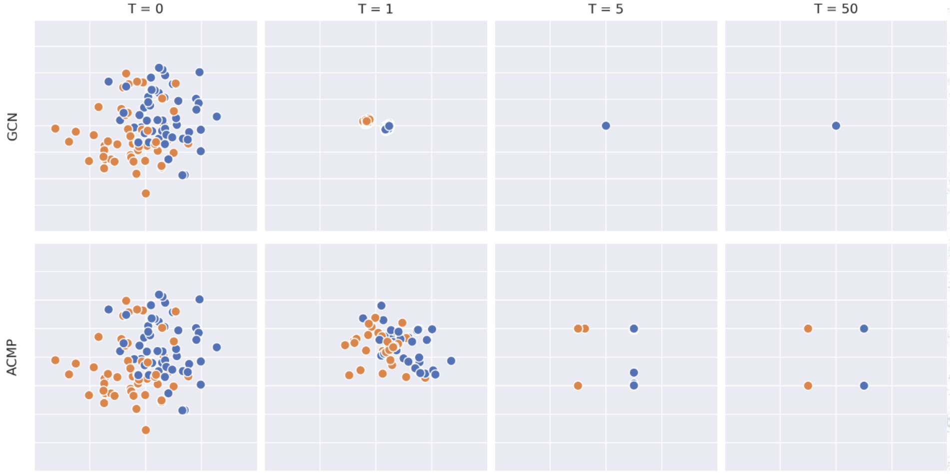
# Allen-Cahn Message Passing (ACMP)

Wang et al. 22

The Allen–Cahn potential introduces repulsive forces that prevent oversmoothing



$$\frac{\partial \mathbf{x}}{\partial t} = -\nabla_{\mathbf{x}}\Phi, \qquad \frac{\partial \mathbf{x}_i}{\partial t} = -\frac{\partial \Phi}{\partial \mathbf{x}_i} = \alpha \sum_{j \in \mathcal{N}_i} a_{i,j}(\mathbf{x}_j - \mathbf{x}_i) + \delta \mathbf{x}_i(1 - \|\mathbf{x}_i\|^2)$$

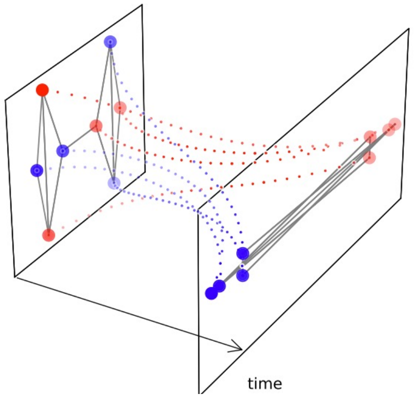



Graph Diffusion

Allen-Cahn double well

# Gradient Flow Framework (GRAFF) Di Giovanni, Rowbottom et al. 22

The gradient flow of a learnable energy can induce repulsive forces that prevent oversmoothing




$$\mathcal{E}^{\mathrm{tot}}(\mathbf{F}) := \frac{1}{2}\sum_{i}\langle \mathbf{f}_i, \boldsymbol{\Omega}\mathbf{f}_i\rangle - \frac{1}{2}\sum_{i,j}\bar{a}_{ij}\langle \mathbf{f}_i, \mathbf{W}\mathbf{f}_j\rangle \equiv \mathcal{E}^{\mathrm{ext}}_{\boldsymbol{\Omega}}(\mathbf{F}) + \mathcal{E}^{\mathrm{pair}}_{\mathbf{W}}(\mathbf{F})$$

Gradient flow gives diffusion with channel mixing and attractive / repulsive forces

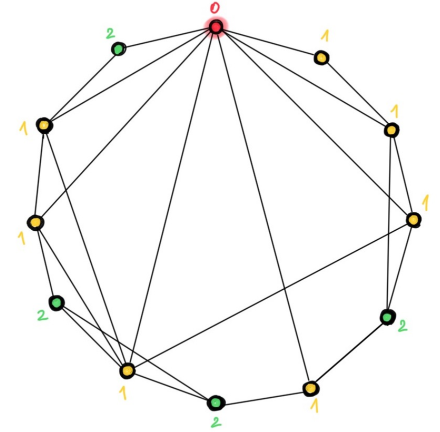
$$\dot{\mathbf{F}}(t) = -\nabla_{\mathbf{F}}\mathcal{E}^{\mathrm{tot}}(\mathbf{F}(t)) = \underbrace{-\mathbf{F}(t)\boldsymbol{\Omega}}_{\text{Damping}} + \underbrace{\bar{\mathbf{A}}\mathbf{F}(t)\mathbf{W}}_{\text{Onsager diffusion with symmetric W}}$$

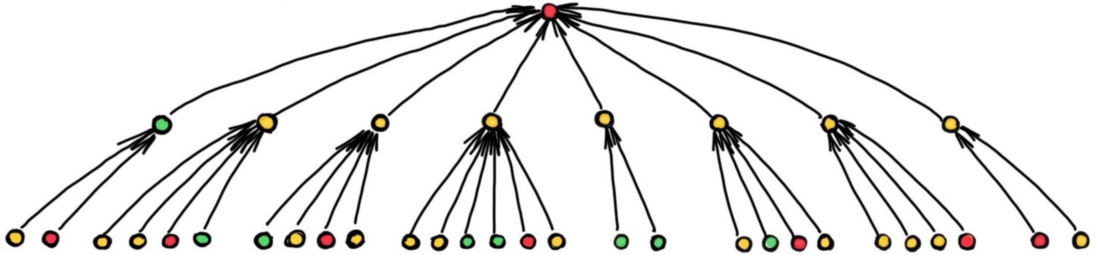
Damping

Onsager diffusion with symmetric W

# Bottlenecks and Oversquashing

# *Over-squashing & Bottlenecks*



In small-world graphs metric ball volume $\mathrm{vol}(B_k) = \sum_{j \in B_k} d_j$ grows exponentially with ball radius $k$

Long-distance dependency + Fast volume growth
= Over-squashing

Alon, Yahav 2020

# *Characterisation of Over-squashing in GNNs*

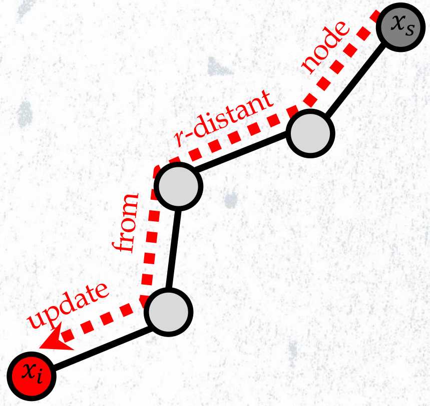
- Multilayer MPNN-type GNN of the form

$$x_i^{(\ell+1)} = \phi_\ell\left(x_i^{(\ell)}, \sum_{j=1}^{n} \hat{a}_{ij}\psi_\ell\left(x_i^{(\ell)}, x_j^{(\ell)}\right)\right)$$

- $|\nabla\phi_\ell| \leq \alpha$ and $|\nabla\psi_\ell| \leq \beta$ for $\ell = 0,1,\dots,L$.

**Lemma 1 (sensitivity):** Let node $s$ be geodesically $d_G(i,s) = r+1$ away from node $i$. Then

$$\left|\frac{\partial h_i^{(r+1)}}{\partial x_s}\right| \leq (\alpha\beta)^{r+1}\left(\widehat{\mathbf{A}}^{r+1}\right)_{is}$$




Over-squashing: small Jacobian $\left|\frac{\partial x_i^{(r+1)}}{\partial x_s}\right|$ leads to poor information propagation

Topping, di Giovanni, et B. 2021

# *Characterisation of Over-squashing in GNNs*

- Multilayer MPNN-type GNN of the form

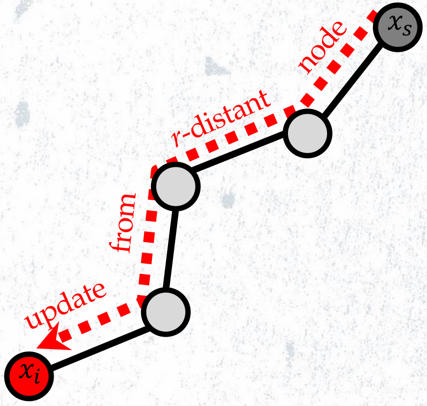
$$x_i^{(\ell+1)} = \phi_\ell\left(x_i^{(\ell)}, \sum_{j=1}^{n} \hat{a}_{ij}\psi_\ell\left(x_i^{(\ell)}, x_j^{(\ell)}\right)\right)$$

- $|\nabla\phi_\ell| \leq \alpha$ and $|\nabla\psi_\ell| \leq \beta$ for $\ell = 0,1,\ldots,L$.

**Lemma 1 (sensitivity):** Let node $s$ be geodesically $d_G(i,s) = r + 1$ away from node $i$. Then

$$\left|\frac{\partial x_i^{(r+1)}}{\partial x_s}\right| \leq (\alpha\beta)^{r+1}\left(\widehat{\mathbf{A}}^{r+1}\right)_{is}$$

It's the graph structure ("bottleneck") to blame!




Over-squashing: small Jacobian $\left|\frac{\partial x_i^{(r+1)}}{\partial x_s}\right|$ leads to poor information propagation

Topping, di Giovanni, et B. 2021

# *Characterisation of Over-squashing in GNNs*

- Multilayer MPNN-type GNN of the form

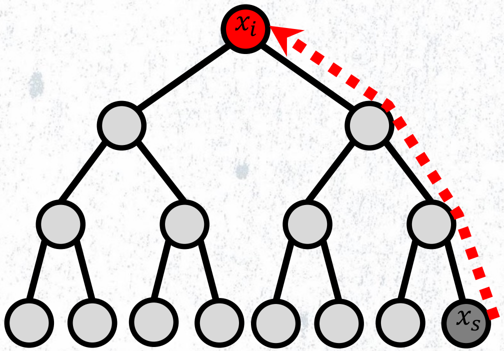
$$x_i^{(\ell+1)} = \phi_\ell\left(x_i^{(\ell)}, \sum_{j=1}^{n} \hat{a}_{ij}\psi_\ell\left(x_i^{(\ell)}, x_j^{(\ell)}\right)\right)$$

- $|\nabla\phi_\ell| \leq \alpha$ and $|\nabla\psi_\ell| \leq \beta$ for $\ell = 0,1,\ldots,L$.

**Lemma 1 (sensitivity):** Let node $s$ be geodesically $d_G(i,s) = r+1$ away from node $i$. Then

$$\left|\frac{\partial x_i^{(r+1)}}{\partial x_s}\right| \leq (\alpha\beta)^{r+1}\left(\hat{\mathbf{A}}^{r+1}\right)_{is}$$

It's the graph structure ("bottleneck") to blame!

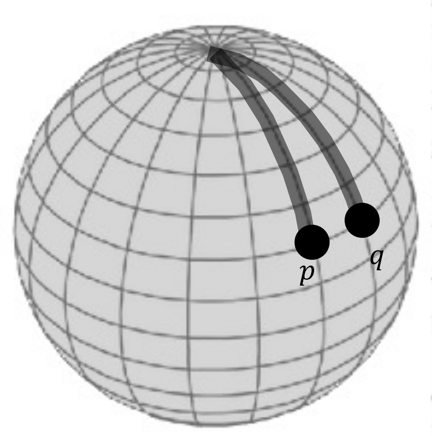



Pathological example: binary tree
$\left(\hat{\mathbf{A}}^{r+1}\right)_{is} = \frac{1}{2}\cdot 3^{-r}$

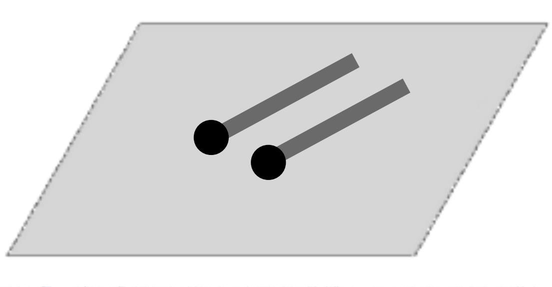
Topping, di Giovanni, et B. 2021

# Graph Curvature

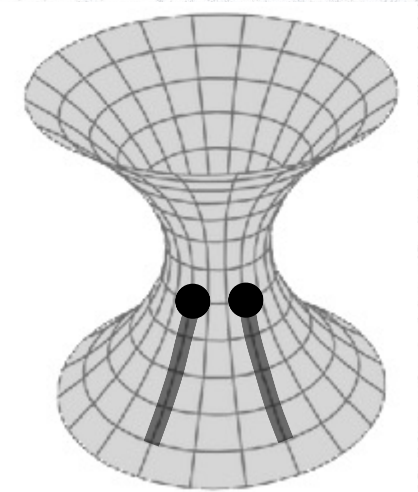
# *Ricci Curvature on Manifolds*




Spherical (>0)



Euclidean (=0)



Hyperbolic (<0)

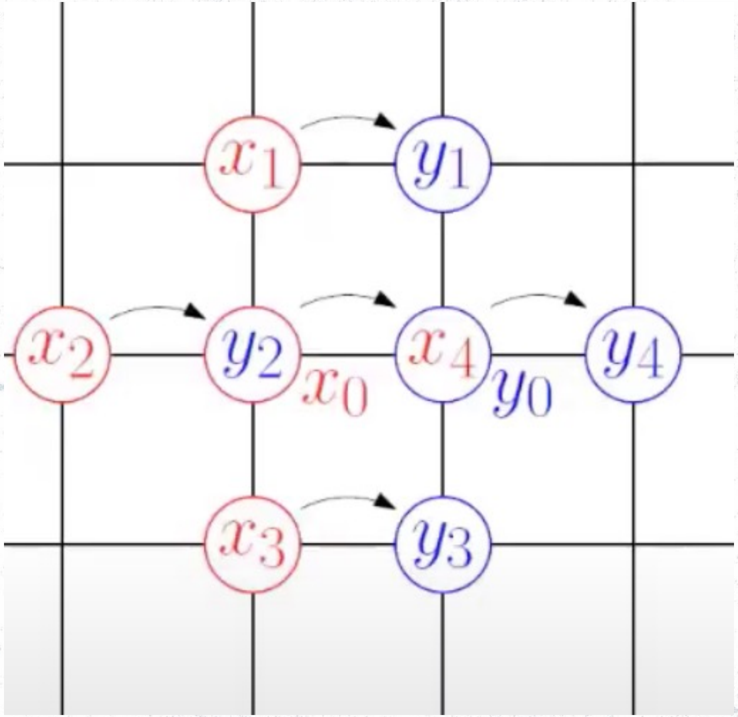
**"geodesic dispersion"**

Ricci 1903

# *Ricci Curvature on Graphs*

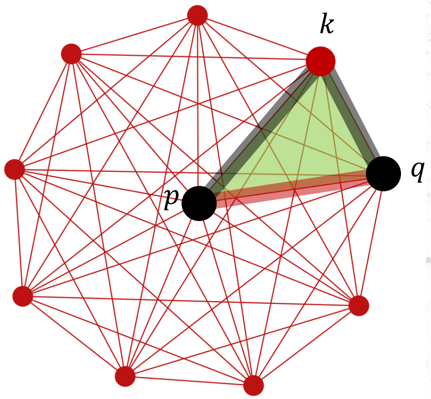
**Sectional curvature defined on edges. Ollivier curvature:**

$$\kappa(x, y) := 1 - \frac{W_1(m_x, m_y)}{d(x, y)},$$

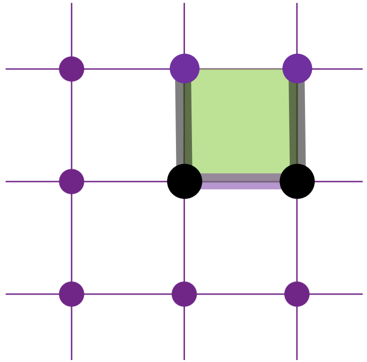


# *Ricci Curvature on Graphs*

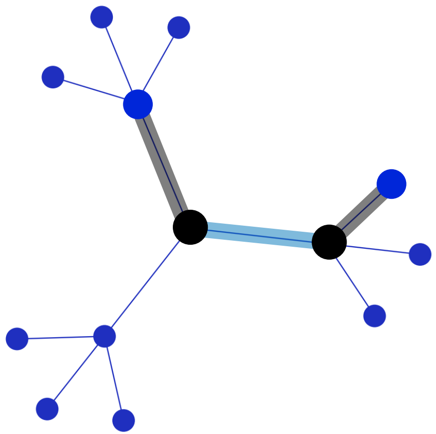
**Sectional curvature defined on edges. Ollivier curvature:** $\kappa(x, y) := 1 - \frac{W_1(m_x, m_y)}{d(x, y)}$,




Clique (>0)



Grid (=0)



Tree (<0)

Forman 2003; Ollivier 2007; Topping, di Giovanni, et B. 2021

# *Balanced Forman Curvature*

**Ollivier curvature is expensive to calculate due to the optimal transport map**

**Balanced Forman Curvature** of edge $i{\sim}j$ in simple unweighted graph $\mathrm{Ric}(i,j)=0$ if $\min\{d_i, d_j\} = 1$ and otherwise

Triangles based at i~j

Max number of 4-cycle based at i~j traversing the same node

$$\mathrm{Ric}(i,j) = \frac{2}{d_i} + \frac{2}{d_j} + 2\frac{|\sharp_\Delta(i,j)|}{\max\{d_i,d_j\}} + \frac{|\sharp_\Delta(i,j)|}{\min\{d_i,d_j\}} + \frac{\gamma_{\max}^{-1}}{\max\{d_i,d_j\}}\left(|\sharp_\square^i(i,j)| + |\sharp_\square^j(i,j)|\right) - 2$$

Degree of i

Neighbours of i forming a 4-cycle based at i~j (w/o diagonals)

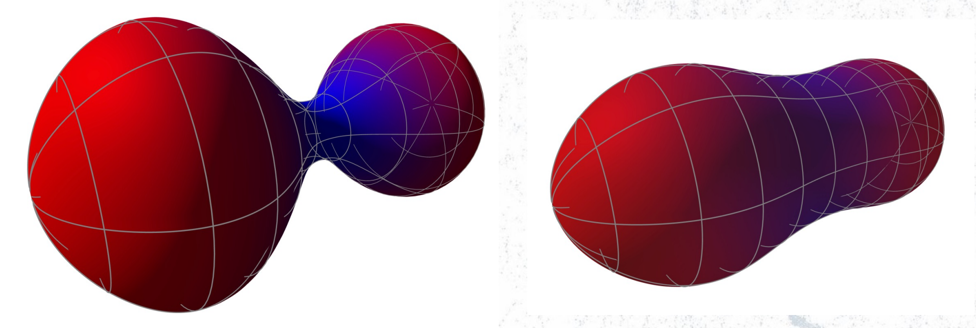
Forman 2003; Topping, di Giovanni, et B. 2021

# Ricci Flow

# *Ricci flow*

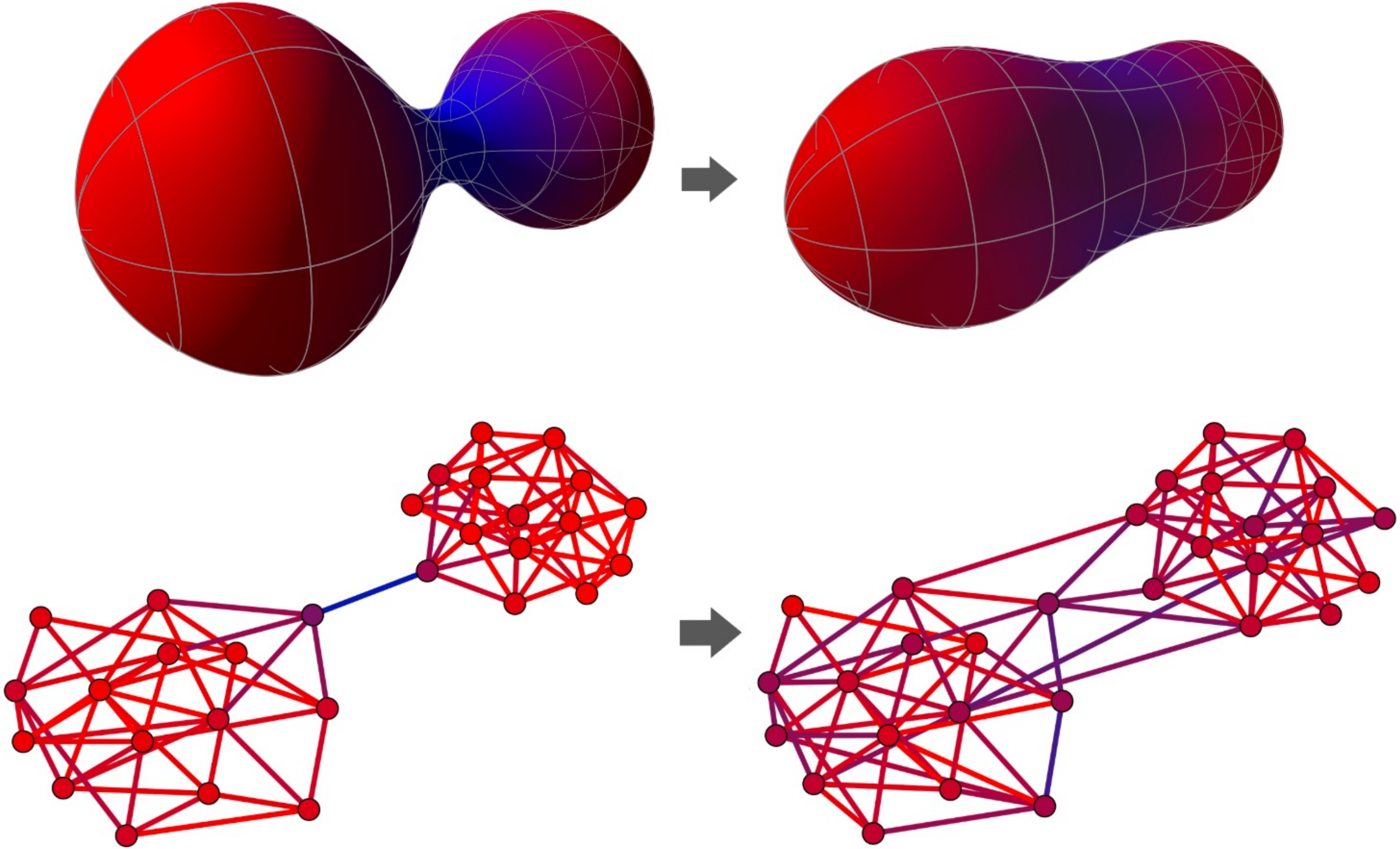
- **Ricci flow:** "diffusion of the Riemannian metric"

$$\frac{\partial g_{ij}}{\partial t} = R_{ij}$$

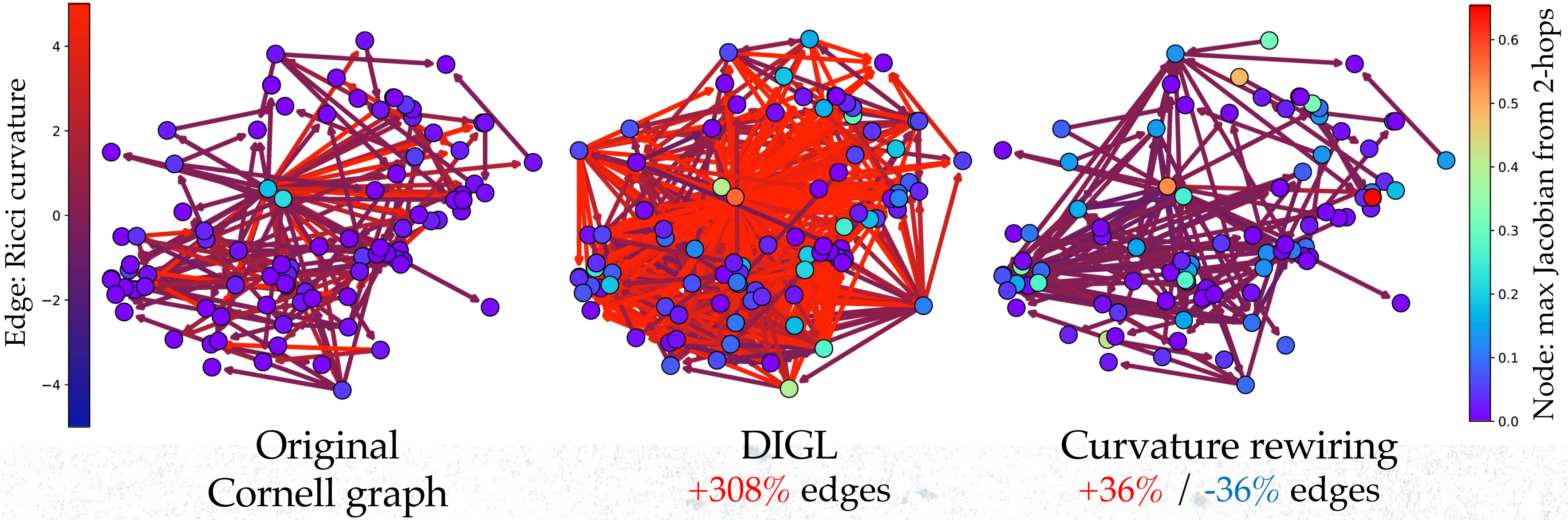


Evolution of a manifold under Ricci flow


Ricci 1903; Hamilton 1988; Perelman 2003



# *Curvature- vs Diffusion-based Rewiring*




Topping, di Giovanni, et B. 2021; Klicpera et al. 2019 (DIGL)

ICLR
International Conference On Learning Representations

APRIL
20
2022

# Announcing the ICLR 2022 Outstanding Paper Award Recipients

YEJIN CHOI / 2022 Conference / awards, ICLR 2022

***By ICLR 2022 Senior Program Chair Yan Liu and Program Chairs Chelsea Finn, Yejin Choi, Marc Deisenroth***

## Outstanding Paper Honorable Mentions

**Understanding over-squashing and bottlenecks on graphs via curvature**

*By Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, Michael M. Bronstein*

For an intro to the paper see: Aleksa Gordic's YouTube channel The AI Epiphany
https://www.youtube.com/c/TheAIEpiphany

# *Summary*

- **GNNs as differential equations – layers and continuous time**
  - **Stability conditions**
  - **Architectures based on numerical solvers**
  - **Control smoothing**
- **Evolving the underlying space and rewiring**
- **Graph Curvature**
  - **Decouping the input and computational graph**
  - **Remove bottlenecks**

# Thank You and Questions

@b_p_chamberlain