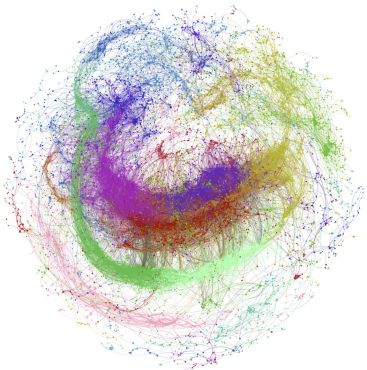
Advection-Diffusion on Graphs:
A Bakry-Émery Laplacian for Spectral Graph Neural Networks



Mia Zosso

Signal Processing Laboratory LTS2, EPFL

EPFL

LTS^2

Paper version

Geometry-Induced Diffusion on Graphs:

A Learnable Weighted Laplacian for Spectral GNNs

Mia Zosso Ali Hariri Victor Kawasaki-Borruat Pierre-Gabriel Berlureau
Pierre Vandergheynst

Same work, updated title in the arXiv version.

Scan for the paper

arXiv:2602.18141

Outline

Motivation: Reduce oversmoothing and oversquashing for spectral GNNs

Goal: Learn how information should propagate

Method: Replace L by a learned weighted Laplacian L_μ

Experiments: Bottlenecks and long-range graph tasks

Conclusion

Motivation

Long-range information propagation is challenging for Graph Neural Networks.

- Local propagation is efficient, but information moves only step by step.
- Increasing the propagation range, either by stacking layers or increasing polynomial order of L:
  - **oversmoothing:** node features become too similar,
  - **oversquashing:** distant information is compressed through bottlenecks,
- Global mechanisms, can improve reach, but are usually computationally expensive.

This work

Improve spectral GNNs by learning the propagation operator L_μ.

Objectives

Objectives

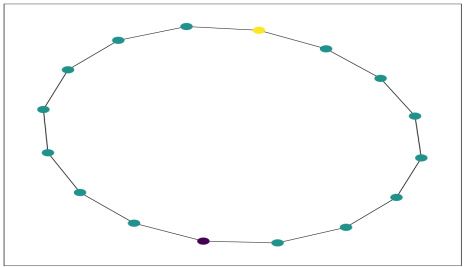
Goal

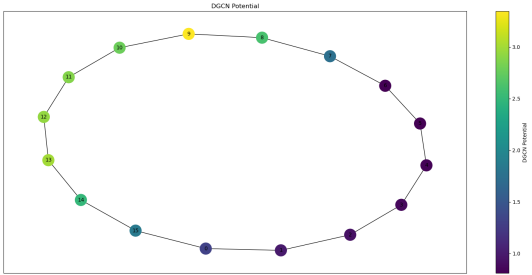
Design a graph propagation operator that:

- learns task-dependent preferred routes,
- improves long-range information propagation,
- keeps the model lightweight.

Use a learned Laplacian L_μ to guide graph diffusion along the existing edges.

Learning preferred routes





Task
Predict the correct class at the query node (purple).

Signal
The class is stored at the answer node (yellow). Two paths connect them: one clean, one noisy.

Learning
The model learns to suppress the noisy path and route information through the clean one.

Method

From diffusion to learned diffusion

Graph diffusion is generated by the graph Laplacian.

$$\partial_t x = -Lx, \qquad x(t) = e^{-tL}x(0).$$

- L controls how signals spread over the graph.
- Replacing L changes the diffusion process.

Idea

Learn a new Laplacian L_μ to guide diffusion on the same graph.

From diffusion to learned diffusion

We learn a positive node density μ_i to modify the cost of local variations.

$$\underbrace{\mathcal{E}(f) = \frac{1}{2}\sum_i \sum_{j\sim i}(f_i - f_j)^2}_{\text{standard energy: uniform local smoothing}} = \langle f, Lf\rangle \quad \Longrightarrow L = D - A$$

$$\underbrace{\mathcal{E}_\mu(f) = \frac{1}{2}\sum_i \mu_i \sum_{j\sim i}(f_i - f_j)^2}_{\text{node-weighted energy}} = \sum_i f_i \sum_{j\sim i} \frac{\mu_i + \mu_j}{2}(f_i - f_j) = \langle f, L_\mu f\rangle \quad \Longrightarrow L_\mu = D_\mu - A_\mu$$

Weighted Laplacian

The learned node-wise density μ induces a weighted graph Laplacian:

$$L_\mu = D_\mu - A_\mu, \qquad (A_\mu)_{ij} = \frac{\mu_i + \mu_j}{2} A_{ij}, \qquad D_\mu = \mathrm{diag}(A_\mu 1).$$

End-to-end construction

$$(X, A) \xrightarrow{\text{small GCN}} \mu \xrightarrow{\text{weighted Laplacian}} L_\mu \xrightarrow{\text{spectral filter}} g_\theta(L_\mu)X$$

X: node features, A: adjacency matrix, μ: learned node density, $g_\theta(L_\mu)$: learned spectral filter.

We can plug it into whatever spectral filter!

Relative spectral speeds

Setup

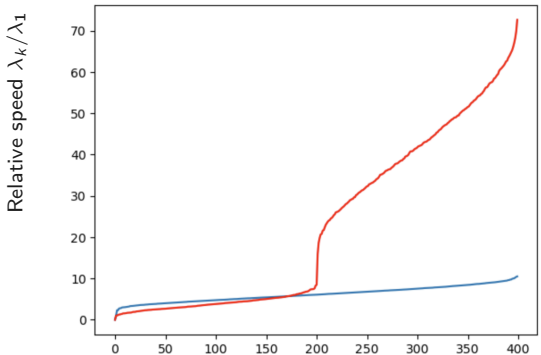
Two-community graph

Meaning

λ_k controls how fast mode k decays under heat diffusion:

$$f_k(t) = e^{-\lambda_k t} f_k(0).$$

The transient regime changes: relevant information can cross the bottleneck before representations collapse.



Blue: standard L. Red: weighted L_μ.

Why does this Laplacian help?

- **Topology-preserving:** no new edges, only reweighting.
- **Oversmoothing:** fewer unnecessary propagation steps.
- **Oversquashing:** stronger propagation across task-relevant bottlenecks.
- **Lightweight:** build L_μ in $O(|E|)$; with ChebNet, filtering costs $O(K|E|)$, hence linear in $|V|$ on sparse graphs.

Experiments

Model instantiations

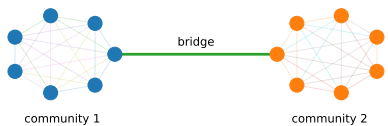
ChebNet-style filters provide a strong sparse backbone for long-range propagation [1].

- μ-**ChebNet:** ChebNet using the learned Laplacian L_μ.
- μ-**Stable-ChebNet:** Stable-ChebNet using the learned Laplacian L_μ.
- **Purpose:** test whether learning the propagation geometry improves long-range graph tasks.

[1] Hariri et al., *Return of ChebNet*, NeurIPS, 2025.

Bottlenecked propagation

Barbell graph



Two dense communities connected by a single bridge edge.

Method	N = 50	N = 70
ChebNet, $K = 9$	0.32 ± 0.39	1.08 ± 0.05
Stable-ChebNet, $K = 9$	0.17 ± 0.11	0.47 ± 0.49
μ-ChebNet, $K = 9$	$\mathbf{0.03 \pm 0.01}$	$\mathbf{0.02 \pm 0.02}$

Method	K	100
ChebNet	$K = 20$	0.87 ± 0.05
Stable-ChebNet	$K = 20$	0.21 ± 0.27
μ-ChebNet (ours)	$K = 15$	$\mathbf{0.02 \pm 0.01}$

MSE, lower is better.

Real-world long-range benchmark

City Networks: nodes are road junctions, edges are road segments.

Method	London	Paris	Shanghai	Los Angeles
ChebNet	39.10 ± 1.79	34.34 ± 0.31	40.00 ± 0.39	41.31 ± 0.48
Stable-ChebNet	40.11 ± 0.20	$\mathbf{38.11 \pm 0.25}$	41.45 ± 0.27	42.17 ± 0.16
μ-**ChebNet (ours)**	$\mathbf{41.88 \pm 0.11}$	35.14 ± 0.12	41.98 ± 0.96	$\mathbf{43.33 \pm 0.33}$
+ **Stability**	41.44 ± 0.07	37.49 ± 0.26	$\mathbf{42.63 \pm 0.70}$	43.10 ± 0.60

Table 1: Accuracy (%), higher is better. All models use the same receptive field $K = 10$, so the comparison isolates the effect of replacing L by L_μ.

Conclusion

Conclusion

Contribution. A learned node density μ induces a task-adaptive Laplacian:

$$(A_\mu)_{ij} = \frac{\mu_i + \mu_j}{2} A_{ij}, \qquad L_\mu = D_\mu - A_\mu.$$

- Preserves topology: no new edges.
- Learns conductances: stronger or weaker propagation.
- Fits spectral GNNs: use L_μ instead of L.
- Improves performance on bottlenecked and long-range tasks.

Takeaway

Learn the diffusion geometry, keep the graph fixed, improve graph propagation.

Thank you

Full paper: 2602.18141

EPFL

LTS^2

Supplementary Material

L_μ is a valid Laplacian

If $\mu_i > 0$, then for any signal f,

$$f^\top L_\mu f = \frac{1}{2} \sum_{i \sim j} \frac{\mu_i + \mu_j}{2} (f_i - f_j)^2 \geq 0.$$

Also,

$$L_\mu 1 = 0.$$

Therefore L_μ is symmetric positive semidefinite and has the constant signal in its kernel.

Topology is preserved

The weighted adjacency is

$$(A_\mu)_{ij} = \frac{\mu_i + \mu_j}{2} A_{ij}.$$

Therefore,

$$A_{ij} = 0 \quad \Longrightarrow \quad (A_\mu)_{ij} = 0.$$

- No new edges are created.
- Only the weights of existing edges are changed.
- The model produces a rewiring-like effect through weighted propagation, not graph rewiring.

Energy-operator correspondence

For any symmetric positive semidefinite bilinear form, once an inner product is fixed, there is a unique symmetric PSD operator L such that

$$\mathcal{E}(f, g) = \langle f, Lg \rangle.$$

For graphs:

$$\mathcal{E}(f, g) = \frac{1}{2} \sum_{i \sim j} (f_i - f_j)(g_i - g_j) = f^\top L g.$$

Thus, changing the energy changes the associated diffusion operator.

$$\mathcal{E}_\mu \quad \Longrightarrow \quad L_\mu \quad \Longrightarrow \quad \partial_t x = -L_\mu x.$$

Continuous vs. graph weighted energy

Continuous case: scalar density weights local gradient energy

$$\mathcal{D}_\mu(f,g) = \int_X \underbrace{\mu(x)}_{\text{scalar density on } X} \underbrace{\langle \nabla f(x), \nabla g(x) \rangle_{T_x X}}_{\text{local gradient product}} \, dx.$$

The graph analogue of "a scalar field on space" is a node signal.

Graph case: node density induces edge conductances

$$\mu_i \text{ on nodes} \quad \rightsquigarrow \quad \mu_{ij} = \frac{\mu_i + \mu_j}{2} \text{ on edges.}$$

$$\mathcal{D}_\mu^{\mathcal{G}}(f,g) = \frac{1}{2} \sum_{i,j} A_{ij} \underbrace{\mu_{ij}}_{\text{edge conductance}} (f_i - f_j)(g_i - g_j).$$

In the continuum, gradients are localized at points. On graphs, gradients live on edges, so node densities must be lifted to edge weights.

Where does $(\mu_i + \mu_j)/2$ come from?

Start from the node-weighted local gradient energy:

$$\mathcal{E}_\mu(f) = \frac{1}{4} \sum_i \mu_i \sum_{j \sim i} (f_i - f_j)^2.$$

For one undirected edge (i, j), there are two endpoint contributions:

$$\frac{1}{4}\mu_i (f_i - f_j)^2 + \frac{1}{4}\mu_j (f_j - f_i)^2.$$

Therefore,

$$\frac{1}{4}(\mu_i + \mu_j)(f_i - f_j)^2 = \frac{1}{2}\frac{\mu_i + \mu_j}{2}(f_i - f_j)^2.$$

So

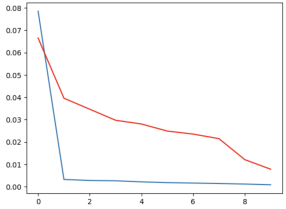
$$\mathcal{E}_\mu(f) = \frac{1}{2} \sum_{i \sim j} \frac{\mu_i + \mu_j}{2} (f_i - f_j)^2.$$

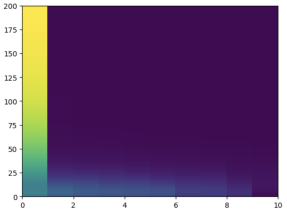
Transient regime

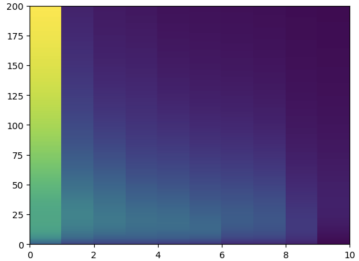
Two-community graph with a bottleneck.

Interpretation:
With L_μ, several singular values remain active for longer.

Message:
The final smoothed state is unchanged, but L_μ opens a transient regime where information can cross the bottleneck before rank collapse.







Top left: singular values of $X(t)$. Top right: L. Bottom: L_μ.

Why “advection-diffusion”?

$$(L_\mu f)_i = \mu_i (Lf)_i - \frac{1}{2} \sum_{j \sim i} \nabla^G_{ij} \mu \, \nabla^G_{ij} f,$$

- The first term is diffusion scaled by the density μ_i.
- The second term couples variations of μ with variations of f.
- This creates an advection-like bias in the propagation.

The density biases diffusion toward selected regions of the graph without introducing an explicit vector field and without modifying the adjacency.

End-to-end training

$$(X, A) \xrightarrow{\text{GCN}_\phi} \mu_\phi \xrightarrow{\text{build } L_\mu} L_\mu \xrightarrow{\text{Chebyshev filter } g_\theta} H \xrightarrow{\text{prediction head}} \widehat{Y}$$

$$H = g_\theta(L_\mu)X = \sum_{k=0}^{K} T_k(\widetilde{L}_\mu)X\Theta_k$$

$$\mathcal{L} = \ell(\widehat{Y}, Y), \qquad (\phi, \theta) \leftarrow (\phi, \theta) - \eta \nabla_{\phi,\theta}\mathcal{L}.$$

- ϕ: parameters of the GCN producing μ.
- $\theta = \{\Theta_k\}$: Chebyshev filter parameters.
- Both are learned jointly from the task loss.

Complexity of the learned Laplacian

1. Computing the node density $(X, A) \xrightarrow{\text{small GCN}} \mu$ The aggregation step is $\hat{A}X$. Since $\hat{A}$ has nonzero entries only on existing edges, the number of nonzero entries satisfies $O(|E|)$. For fixed depth and fixed hidden widths $(|E|)$. If the graph is sparse, then $O(|E|) = O(|V|)$ So, computing μ is linear in the number of nodes on sparse graph families.

2. Building the weighted Laplacian

For every edge (i, j), compute $(A_\mu)_{ij} = \frac{\mu_i + \mu_j}{2} A_{ij}$.

This requires one operation per edge: $\text{cost}(A_\mu) = O(|E|)$.

Then $D_\mu = \text{diag}(A_\mu 1)$ is also computed by summing edge weights: $\text{cost}(D_\mu) = O(|E|)$. Therefore, $\text{cost}(L_\mu) = O(|E|)$.

Complexity of the learned Laplacian

3. Spectral filtering

For a Chebyshev filter of order K,

$$g_\theta(L_\mu)X = \sum_{k=0}^{K} \theta_k T_k(L_\mu)X.$$

The recurrence uses K sparse matrix-feature multiplications, so

$$\text{cost} = O(K|E|).$$

Total

$O(|E|) + O(K|E|) = O((K+1)|E|)$.

On sparse graphs, $|E| = O(|V|)$, hence $O((K+1)|V|) \simeq O(K|V|)$.

What is ChebNet?

A spectral graph filter is naturally written in the eigenbasis of the Laplacian:

$$L = U\Lambda U^\top, \qquad X' = U g_\theta(\Lambda) U^\top X.$$

This requires computing the eigenvectors U, which is expensive for large graphs.

ChebNet avoids this by parameterizing the filter as a Chebyshev polynomial:

$$g_\theta(\lambda) \approx \sum_{k=0}^{K} \theta_k T_k(\widetilde{\lambda}).$$

Therefore,

$$X' = g_\theta(\widetilde{L})X \approx \sum_{k=0}^{K} T_k(\widetilde{L}) X \Theta_k.$$

Same spectral idea, but implemented directly in the node domain.

What is Stable-ChebNet?

ChebNet. Larger K increases the receptive field, but high-order filters can lead to unstable propagation.

Stable-ChebNet. Interpret the Chebyshev update as a dynamical system:

$$\frac{dX(t)}{dt} = \sum_{k=0}^{K} T_k(L)X(t)W_k.$$

Then enforce stable dynamics by antisymmetrizing the weights:

$$W_k - W_k^\top.$$

After Euler discretization:

$$X^{(\ell+1)} = X^{(\ell)} + \epsilon \sum_{k=0}^{K} T_k(L)X^{(\ell)}(W_k - W_k^\top - \gamma I).$$

Graph Property Prediction

Model	Diameter ↓	SSSP ↓	Eccentricity ↓
GCN	0.7424 ± 0.0466	0.9499 ± 0.0001	0.8468 ± 0.0028
GAT	0.8221 ± 0.0752	0.6951 ± 0.1499	0.7909 ± 0.0222
GraphSAGE	0.8645 ± 0.0401	0.2863 ± 0.1843	0.7863 ± 0.0207
GIN	0.6131 ± 0.0990	-0.5408 ± 0.4193	0.9504 ± 0.0007
GCNII	0.5287 ± 0.0570	-1.1329 ± 0.0135	0.7640 ± 0.0355
DGC	0.6028 ± 0.0050	0.1483 ± 0.0231	0.8261 ± 0.0032
GRAND	0.6715 ± 0.0490	-0.0942 ± 0.3897	0.6602 ± 0.1393
A-DGN w/ GCN backbone	0.2271 ± 0.0804	-1.8288 ± 0.0607	0.7177 ± 0.0345
ChebNet	-0.1517 ± 0.0343	-1.8519 ± 0.0539	-1.2151 ± 0.0852
Stable-ChebNet	-0.2477 ± 0.0526	-2.2111 ± 0.0160	**-2.1043 ± 0.0766**
μ-**ChebNet (ours)**	-0.2075 ± 0.0634	-2.3149 ± 0.0301	-1.8740 ± 0.0597
+ Stability	**-0.3179 ± 0.0182**	**-2.3217 ± 0.0330**	**-2.0338 ± 0.0571**

Mean test $\log_{10}(\mathrm{MSE})$, lower is better.

Ogbn-proteins

Task. Multi-label protein function prediction on a protein-protein interaction graph.

Model	**ROC-AUC** ↑
MLP	72.04 ± 0.48
GCN	72.51 ± 0.35
ChebNet	77.55 ± 0.43
Stable-ChebNet	79.55 ± 0.34
SGC	70.31 ± 0.23
GCN-NSAMPLER	73.51 ± 1.31
GAT-NSAMPLER	74.63 ± 1.24
SIGN	71.24 ± 0.46
NodeFormer	77.45 ± 1.15
SGFormer	79.53 ± 0.38
SPEXPFORMER	$\mathbf{80.65 \pm 0.07}$
μ-ChebNet (ours)	79.36 ± 0.41

Table 2: μ-ChebNet remains competitive with stronger architectures while relying on sparse graph operations.

Ogbn-proteins: interpretation

- μ-ChebNet reaches 79.36% ROC-AUC.
- It outperforms standard MPNN baselines and remains competitive with Stable-ChebNet and graph transformers.
- SPEXPHORMER obtains the best score, but uses a more involved attention-based mechanism.
- Our method keeps sparse graph operations:

$$\text{learn } \mu \quad \rightarrow \quad O(m) \text{ construction of } L_\mu \quad \rightarrow \quad O(Km) \text{ Chebyshev filtering.}$$

Open questions and future directions

- **Architectures:** test L_μ beyond ChebNet-style polynomial filters.
- **Parameterization:** study alternative ways (other than GCN) to learn and regularize the density μ.
- Understand the probably non-trivial interaction with stable-ChebNet, as it yields mixed gains.