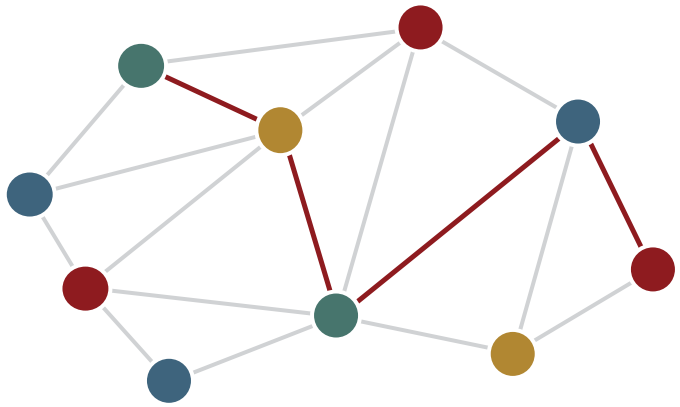
Learnable Graph Filters
Heat kernels, polynomials, and GNNs



September 8

Zhiyang Wang · Electrical & Systems Engineering · WashU

Class 5 · Heat kernels and learnable graph filters

Paper presentation and peer feedback

ESE 5291 20 minutes per presentation + 5-10 mins Q&A and discussions

Preparing your presentation

Choose a paper or propose the paper in the shared sign-up sheet. Present alone or in a pair.

Explain the problem, contribution, core method or theory, and key results. Connect to course concepts.

Critique limitations and reproducibility. End with takeaways, future directions, and discussions.

Use clear, cited slides. Rehearse to fit the time. Code execution is optional.

Completing the feedback sheet

After Q&A, rate each criterion from 1–5:

Problem and contribution, technical understanding, results and evidence, critical evaluation, communication and timing, Q&A responses.

Add a specific strength, an actionable improvement, and a discussion question.

Evaluation keeps you focused and helps improve your own presentation skills. Respect each other's preparation and give honest, thoughtful feedback.

Peer ratings have limited influence on presenters' scores.

Your thoughtful feedback counts toward your in-class engagement and discussion participation.

Three questions organize the lecture

01 **HOW DOES HEAT MOVE?**

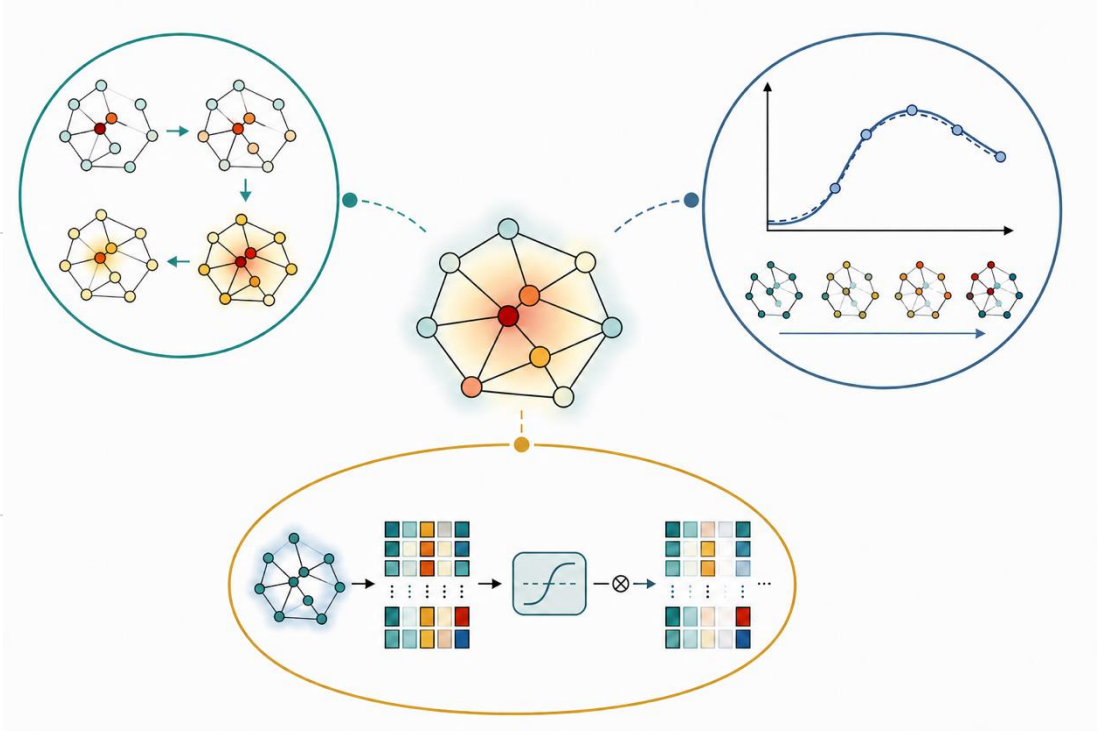
Solve graph diffusion in the vertex and spectral domains.

02 **HOW DO WE DESIGN FILTERS?**

Match a target response with a stable, local polynomial.

03 **HOW DO FILTERS BECOME GNNs?**

Turn filter banks into multi-feature nonlinear layers.



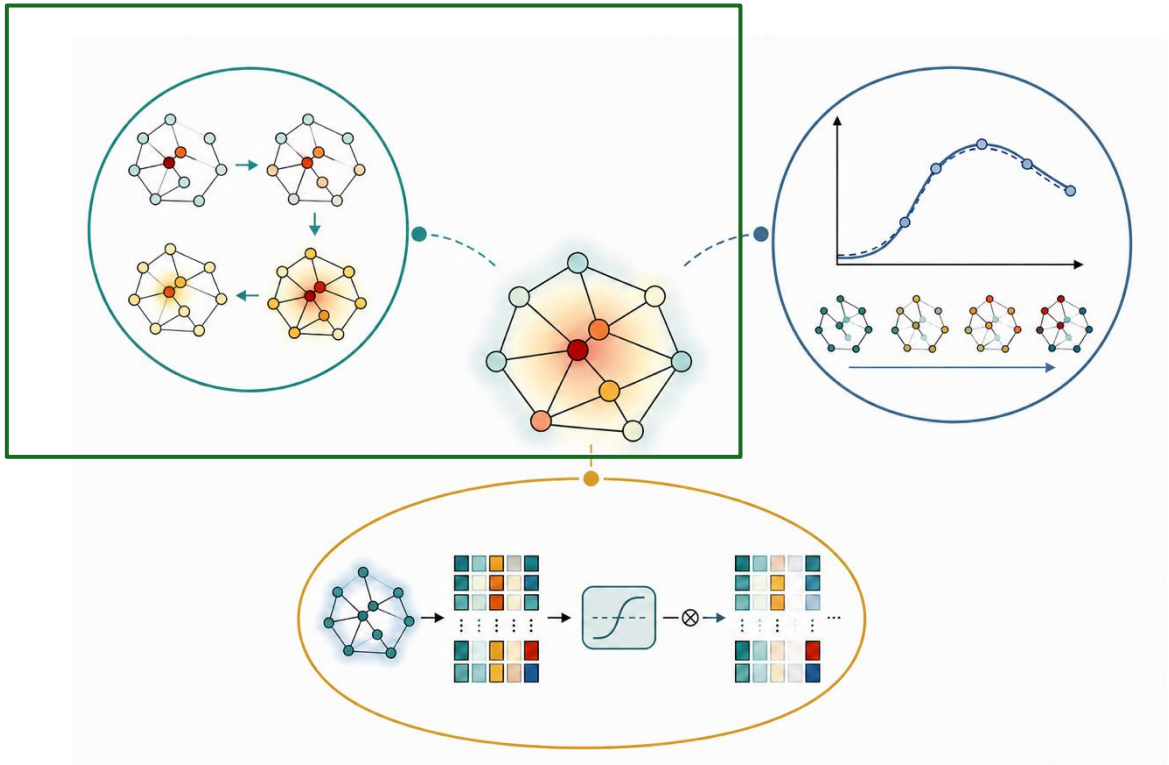
PART I

How does heat move over a graph?

01 Neighbors exchange heat; the Laplacian writes the law $dx/dt = -Lx$

02 The solution is the heat kernel $H_t(L)$ — a low-pass graph filter

03 Diffusion time t widens the spread and smooths detail — the limit is consensus



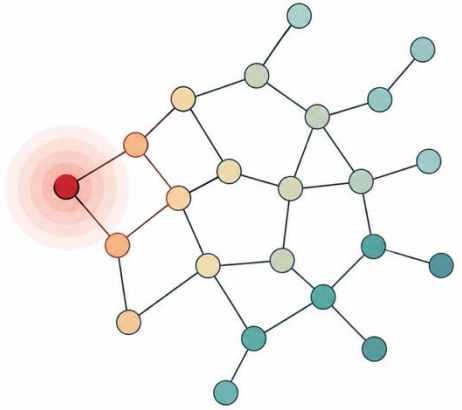
Heat diffusion is the prototype graph filter.

Heat diffusion turns topology into smoothing

PHYSICS · HEAT FLOWS

Each node exchanges heat with neighbors.

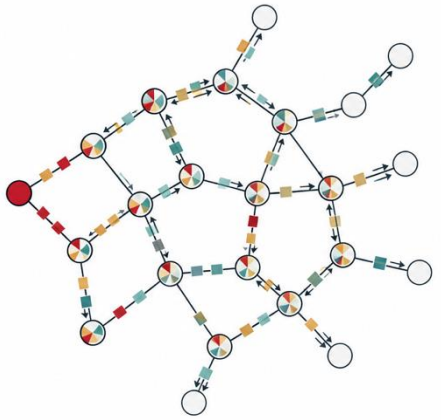
Large disagreements decay fastest.
The Laplacian defines the coupling.



LEARNING · FEATURES PROPAGATE

Messages follow the same edges.

Diffusion averages noisy local evidence.
Scale controls how far information travels.



Heat diffusion is the canonical graph low-pass process.

The graph heat equation has a closed-form solution

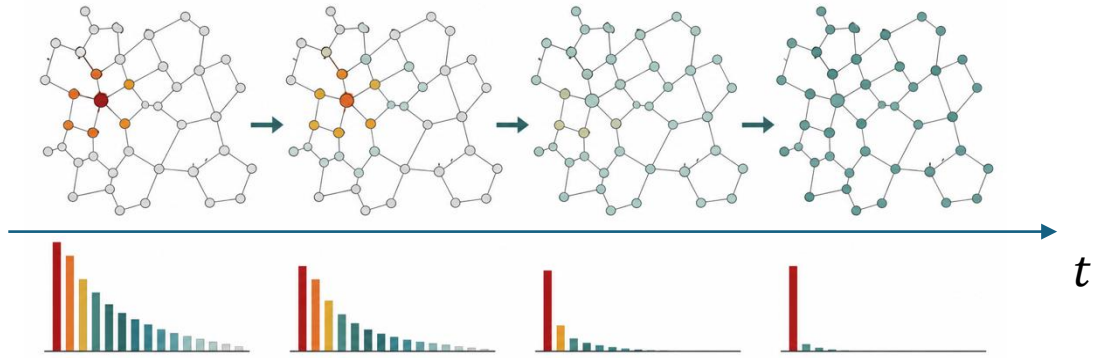
Let L be a graph Laplacian and $\boldsymbol{x}(t)$ the signal evolving on the nodes, the rate of signal change:

$$\frac{d\boldsymbol{x}(t)}{dt} = -\boldsymbol{L}\boldsymbol{x}(t), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0 \qquad\qquad \frac{dx_i(t)}{dt} = -\sum_{j \in \mathcal{N}_i} w_{ij}\left(x_i(t) - x_j(t)\right)$$

The solution is a linear operator applied to the initial condition:

$$\boldsymbol{x}(t) = e^{-t\boldsymbol{L}}\boldsymbol{x}_0 = H_t(\boldsymbol{L})\boldsymbol{x}_0 = (\boldsymbol{I} - t\boldsymbol{L} + \frac{t^2}{2}\boldsymbol{L}^2 - \frac{t^3}{6}\boldsymbol{L}^3 + \cdots + \frac{(-t)^k}{k!}\boldsymbol{L}^k + \cdots)\boldsymbol{x}_0$$

- The matrix exponential $H_t(\boldsymbol{L})$ is the graph heat kernel.
  - Smoothing: the average is preserved and sharp node-to-node differences are damped.



Diffusion time t indexes a continuous family of graph filters.

Three views reveal how the heat kernel works

01 **VERTEX DOMAIN**

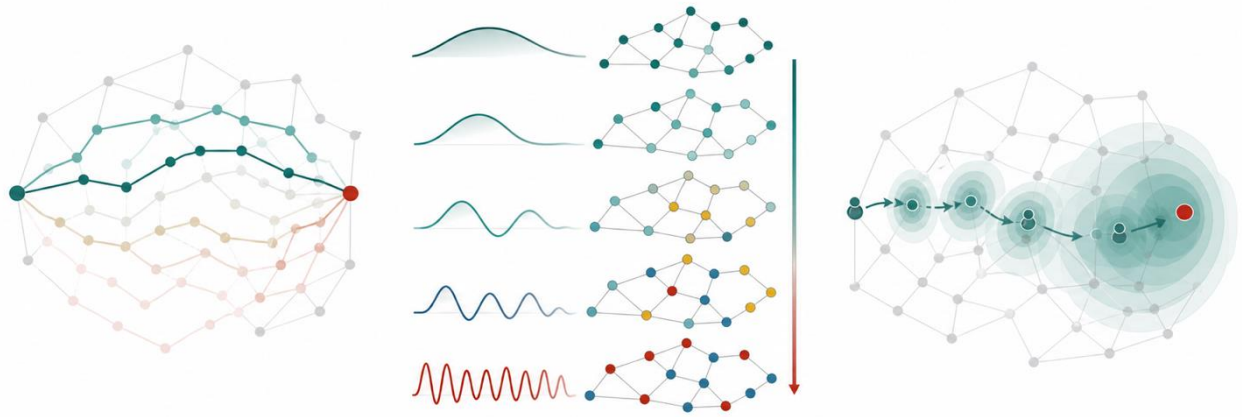
Mix paths of every length, with longer paths weighted progressively less.

02 **SPECTRAL DOMAIN**

Scale Laplacian mode v_i by $exp(-t\lambda_i)$;
high frequencies decay faster.

03 **RANDOM-WALK VIEW**

For a suitable generator, heat mass describes transition probabilities.



Heat kernel is an exponential low-pass graph filter

$$\boldsymbol{L} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}, \qquad H_{\mathrm{t}}(\boldsymbol{L}) = \boldsymbol{V}exp(-t\boldsymbol{\Lambda})\boldsymbol{V}^{\mathrm{T}}$$

Every graph Fourier component evolves independently under diffusion.

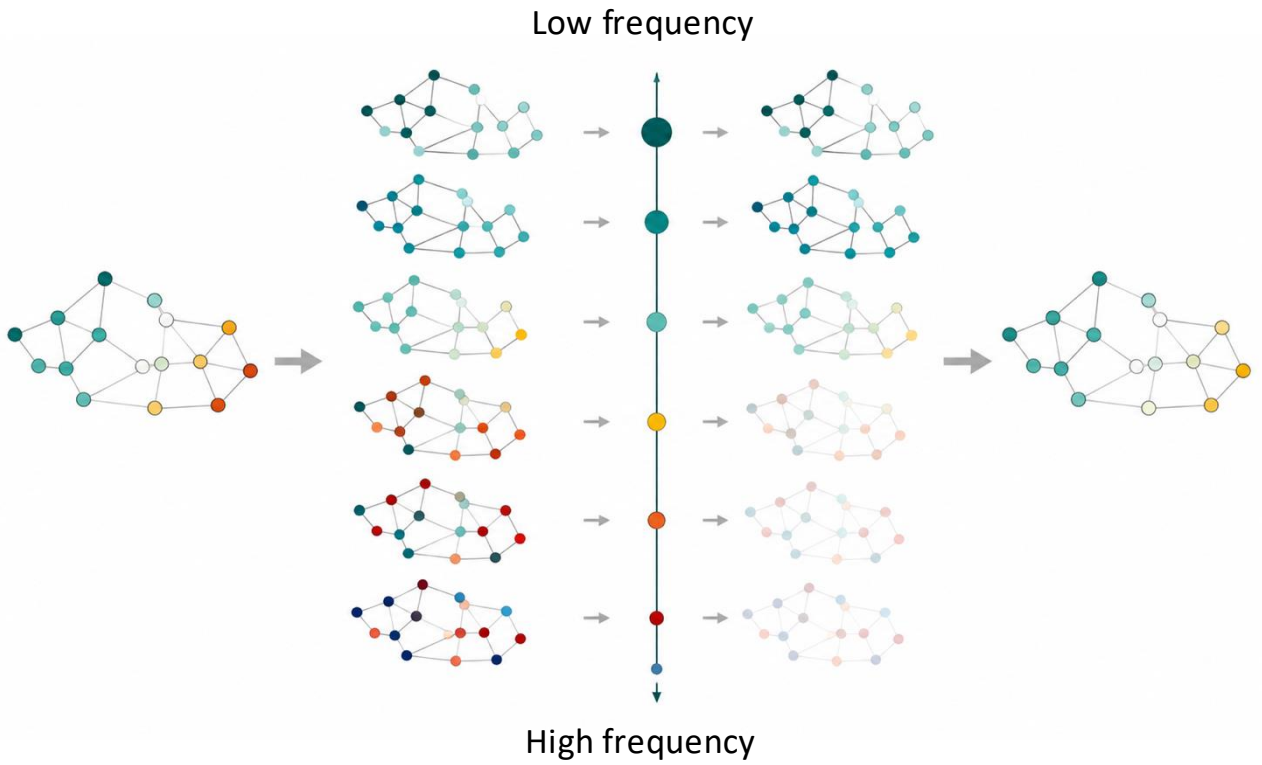
SPECTRAL REPRESENTATION

$$\hat{\boldsymbol{y}} = e^{-t\boldsymbol{\Lambda}}\hat{\boldsymbol{x}}, \qquad \hat{y}_i = e^{-t\lambda_i}\hat{x}_i$$

FREQUENCY RESPONSE FUNCTION

$$h_{\mathrm{t}}(\lambda) = exp(-t\lambda)$$

$h_{\mathrm{t}}(0) = 1$. *For $\lambda > 0$ the gain decreases monotonically, and larger-λ modes vanish faster.*



Heat smooths because its response is strongest at low graph frequencies.

Diffusion time sets the spectral smoothing scale

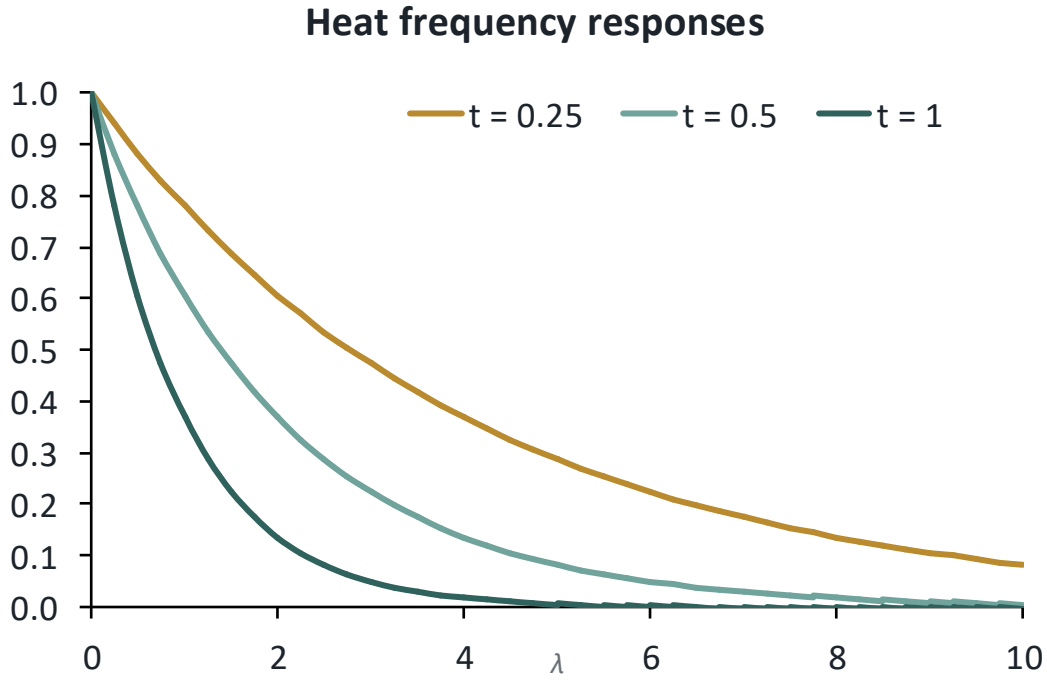
- Small t: preserve local detail
- Large t: suppress high frequencies
- Every response satisfies $h_t(0) = 1$

$$h_t(\lambda) = exp(-t\lambda)$$

TIME-FREQUENCY DUALITY

larger t ⇔ narrower low-pass response

The same parameter that increases diffusion radius also reduces spectral bandwidth.



Time is the single knob that trades detail for smoothness.

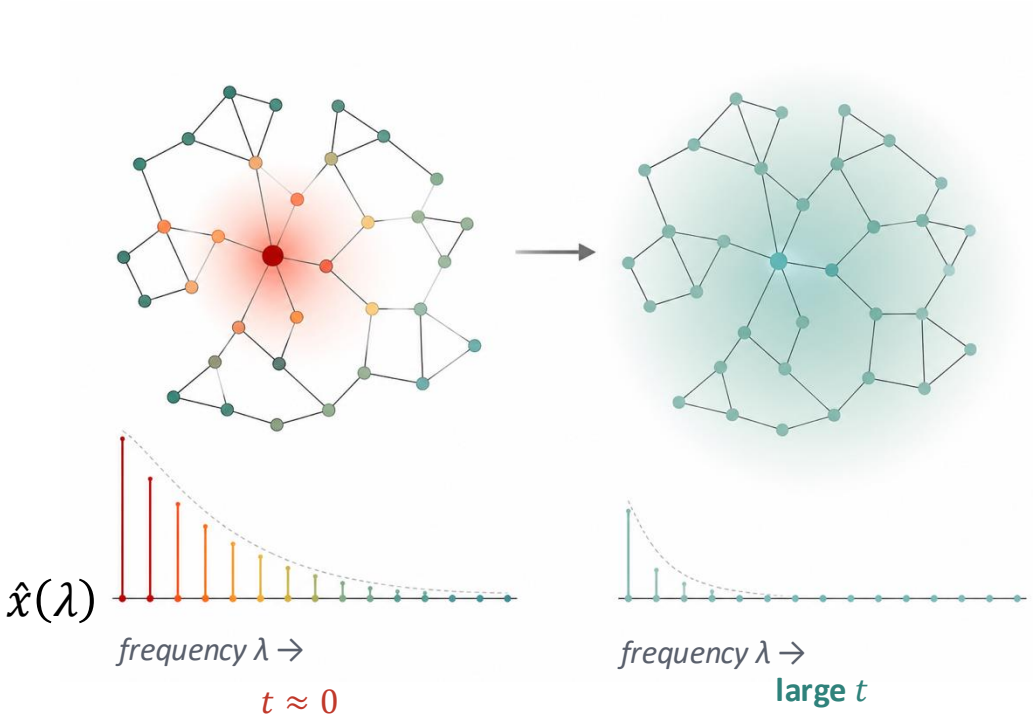
Diffusion moves detail toward consensus

Start from a localized signal x_0.

$$\boldsymbol{x}(t) = exp(-t\boldsymbol{L})\boldsymbol{x}_0$$

- At $t \approx 0$, heat remains near the source and sharp graph-frequency content is visible.
- As t grows, energy spreads through more paths and node values become smoother.

On a connected graph, only the zero-frequency component survives as $t \to \infty$.



Diffusion expands the spatial footprint while contracting spectral variation.

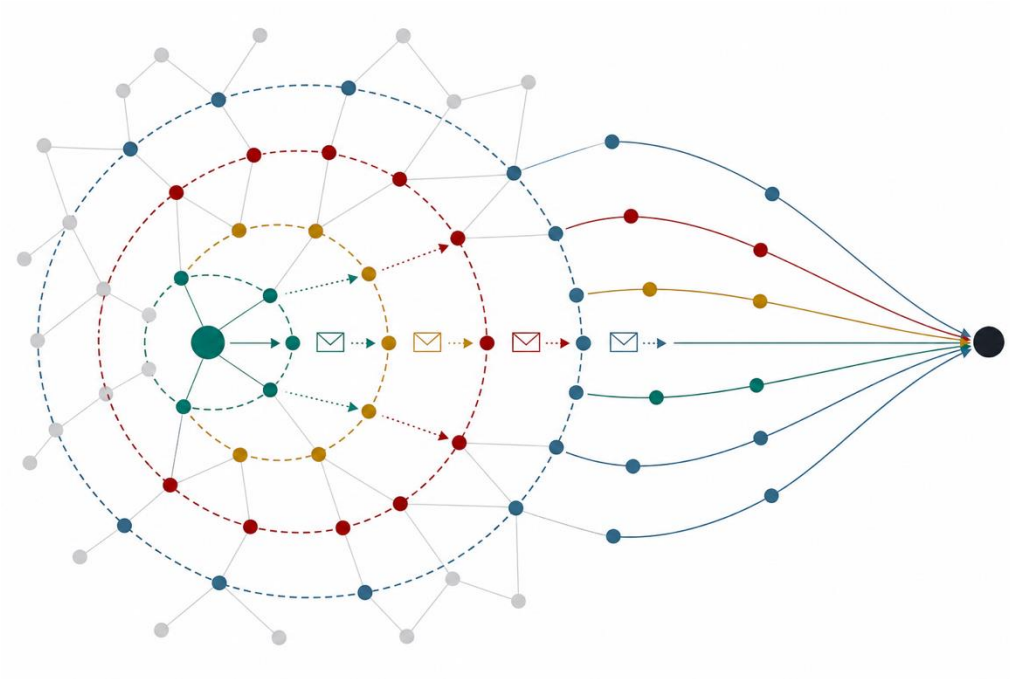
A signal can be narrow in space or narrow in frequency, but not both

Local polynomials implement heat diffusion

Approximate the matrix exponential with a K-term polynomial:

$$e^{-t\boldsymbol{L}}\boldsymbol{x} \approx \sum_{k=0}^{K-1} h_{\mathrm{k}}\boldsymbol{L}^{\mathrm{k}}\boldsymbol{x}$$

- Each multiplication by $\boldsymbol{L}$ requires one neighbor exchange.
- Compute $u_0 = x$, $u_{\mathrm{k}} = \boldsymbol{L}u_{\mathrm{k}-1}$, and accumulate $z = \sum_{\mathrm{k}} h_{\mathrm{k}}u_{\mathrm{k}}$.
- After $K-1$ rounds, z depends only on nodes within $K-1$ hops.



Note *$exp(-t\boldsymbol{L})$ is an infinite polynomial in $\boldsymbol{L}$; finite-K message passing is its truncation—exact as $K \to \infty$, near-exact when $t\|\boldsymbol{L}\|$ is small.*

Polynomial approximation removes eigendecomposition and enables distributed execution.

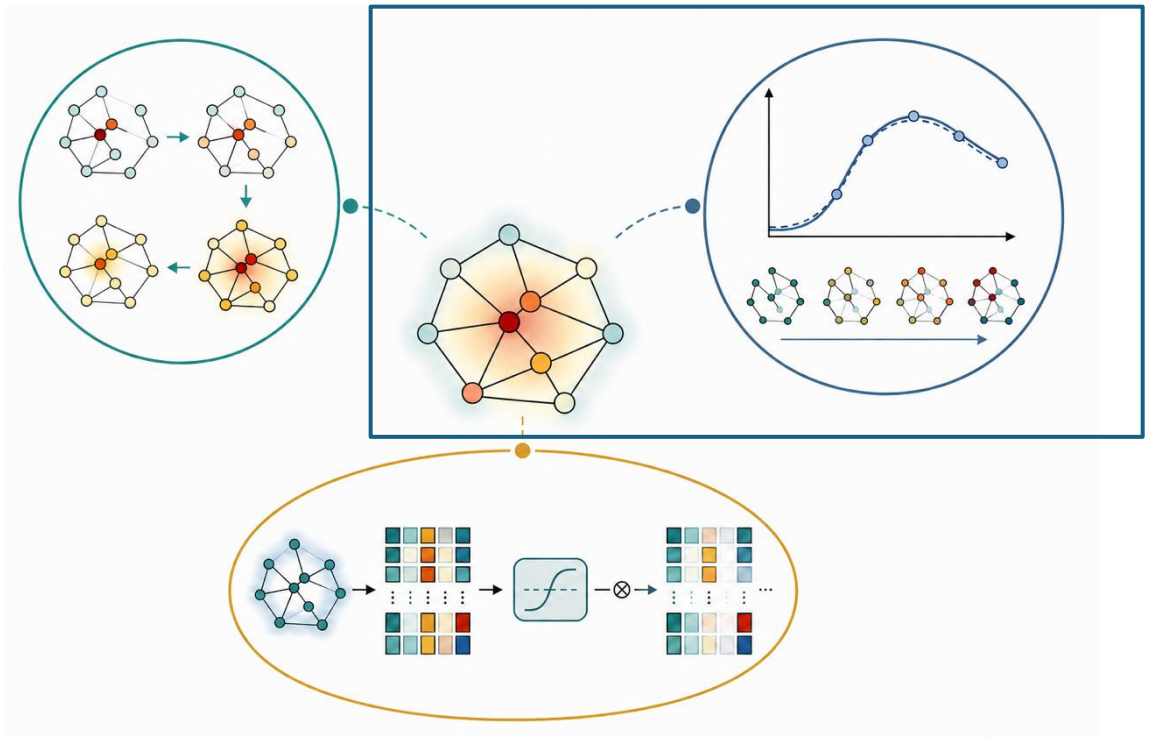
PART II

How do we design graph filters?

01 Truncate diffusion: polynomials in S run on $K-1$ hops of local exchange

02 Free the taps: fit the response $\hat{\mathrm{h}}(\lambda)$ to a target by weighted least squares

03 Spend order wisely: K buys steepness; Chebyshev keeps the fit local



Filter design is approximation theory on the graph spectrum.

Filter design resolves three choices

01 **TARGET RESPONSE**

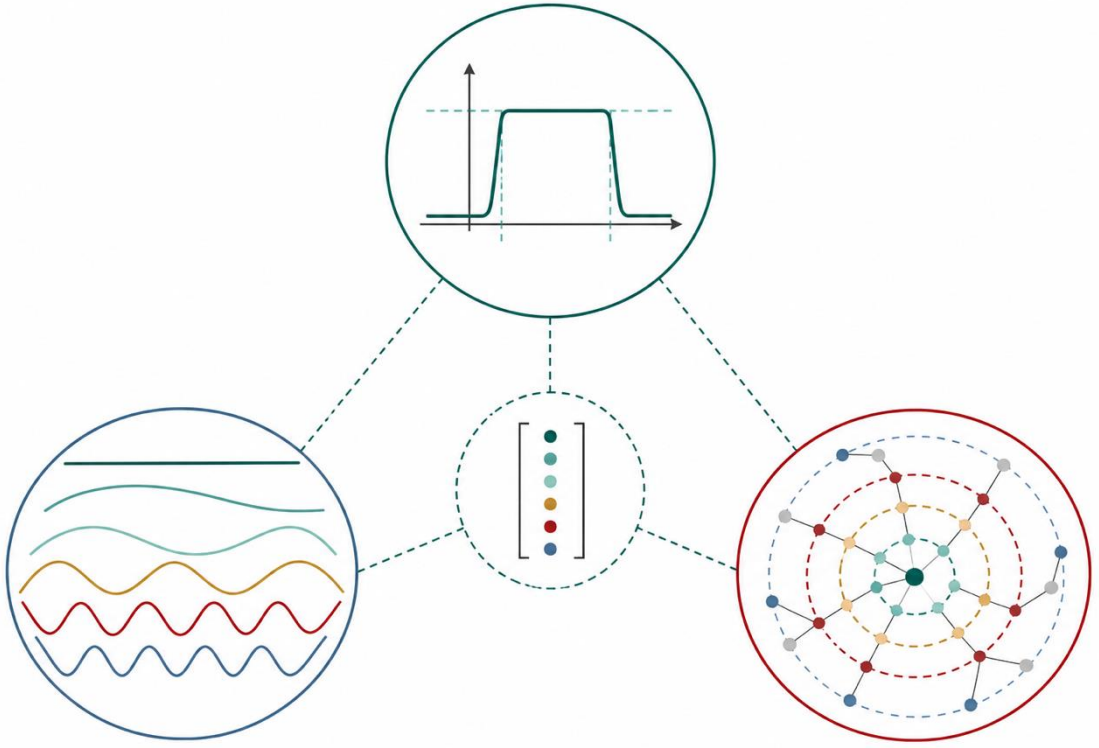
Specify which graph frequencies pass, stop, or receive emphasis.

02 **POLYNOMIAL BASIS**

Choose monomials, Chebyshev polynomials, or another local basis.

03 **ORDER AND CONSTRAINTS**

Balance approximation error, locality, conditioning, and robustness.



Design matches a polynomial to a target response

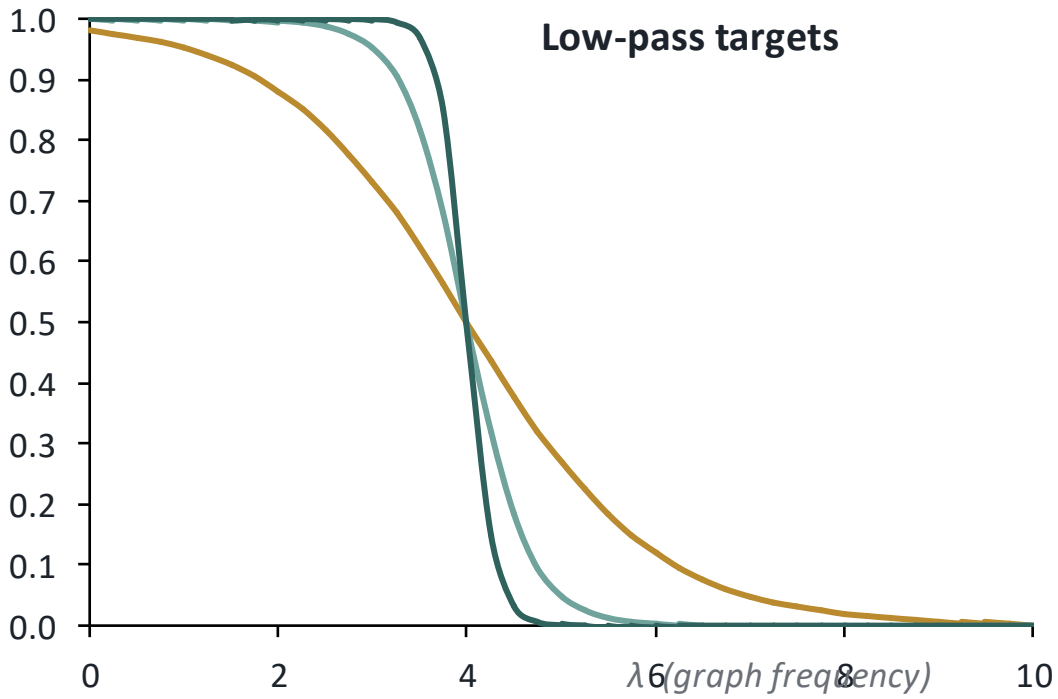
- Choose spectral samples λ_i
- Set desired gains $d_i = g(\lambda_i)$
- Fit $h = [h_0, \ldots, h_{K-1}]$

$$\hat{h}(\lambda) = \sum_{k=0}^{K-1} h_k \lambda^k$$

RESPONSE-MATCHING OBJECTIVE

$$min_h \sum_i w_i \left| g(\lambda_i) - \hat{h}(\lambda_i) \right|^2$$

Weights emphasize important bands; regularization controls tap magnitude and sensitivity.



Filter design is approximation theory on the graph spectrum.

The designer controls fit, order, and regularization

CONTROLLED · ARCHITECTURE

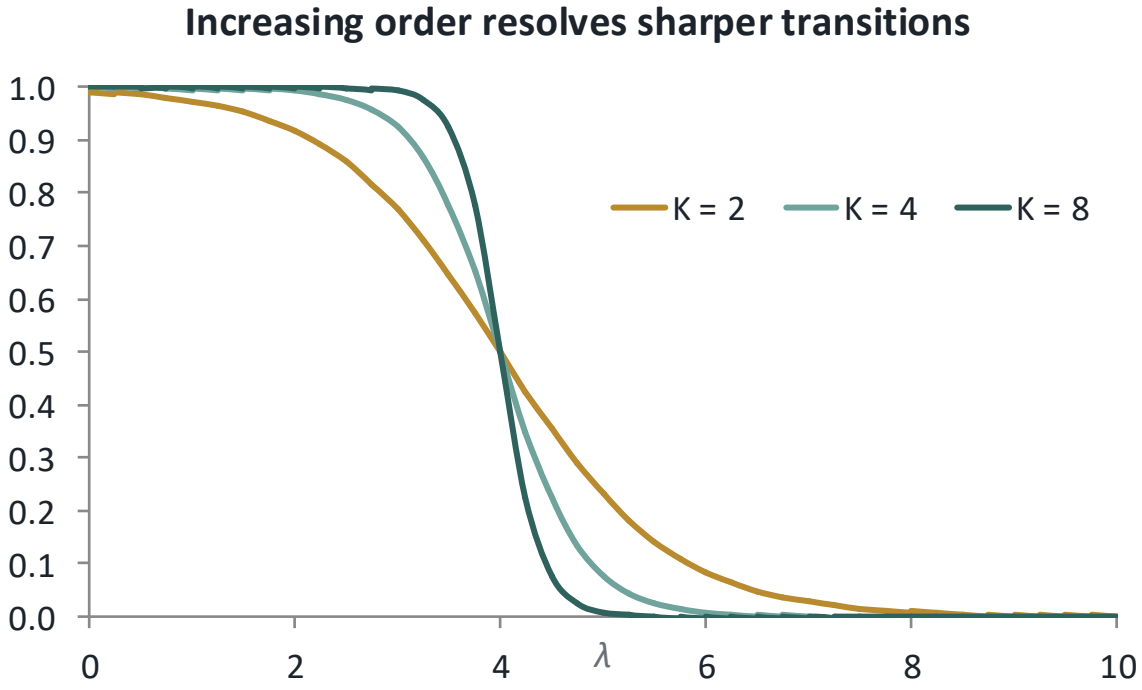
Order K sets receptive field.

Basis sets numerical conditioning and parameterization .
Weights and penalties shape the fit.

PROBLEM-DEPENDENT · SPECTRUM

Eigenvalue density sets where accuracy matters.

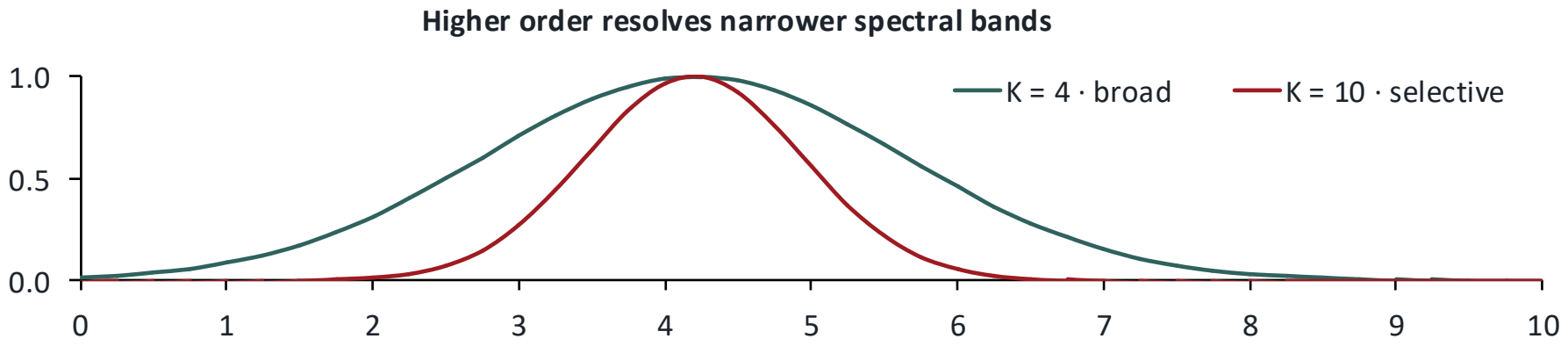
Desired bands come from the task.
Deployment sets the compute budget.



A useful filter is accurate where needed and feasible where deployed.

Order trades spectral selectivity for locality

- K coefficients fit finer spectral detail and narrower transitions
  - A degree-$(K-1)$ polynomial reaches nodes up to $K-1$ hops away
  - Higher order costs more communication and can amplify numerical error



Order is both a spectral-resolution parameter and a communication radius.

Least-squares design becomes a linear system

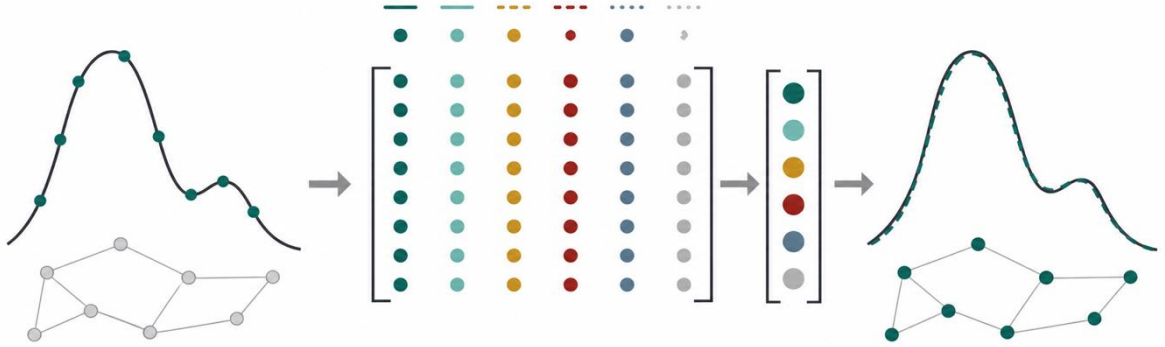
Sample the target $g(\lambda)$ at m spectral points and build a Vandermonde matrix:

$$\boldsymbol{\Psi}_{ik} = \lambda_i^{\,k}, \qquad \boldsymbol{\Psi h} \approx \boldsymbol{g}$$

- The weighted regularized problem has a closed-form normal equation:

$$\min_{\mathbf{h}} \|\boldsymbol{W}^{1/2}(\boldsymbol{\Psi}\mathbf{h} - \mathbf{g})\|_2^2 + \mu\, \mathbf{h}^{\mathrm{T}} \boldsymbol{R} \mathbf{h}$$

$$(\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{\Psi} + \mu\boldsymbol{R})\boldsymbol{h} = \boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{g}$$



$$\boldsymbol{\Psi} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^{\,2} & \cdots & \lambda_1^{\,K-1} \\ 1 & \lambda_2 & \lambda_2^{\,2} & \cdots & \lambda_2^{\,K-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_m & \lambda_m^{\,2} & \cdots & \lambda_m^{\,K-1} \end{bmatrix}, \qquad \boldsymbol{R} = \boldsymbol{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$\boldsymbol{W}$ = diagonal weights w_i (which bands matter) · $\mu\boldsymbol{R}$ penalizes large taps for stability.

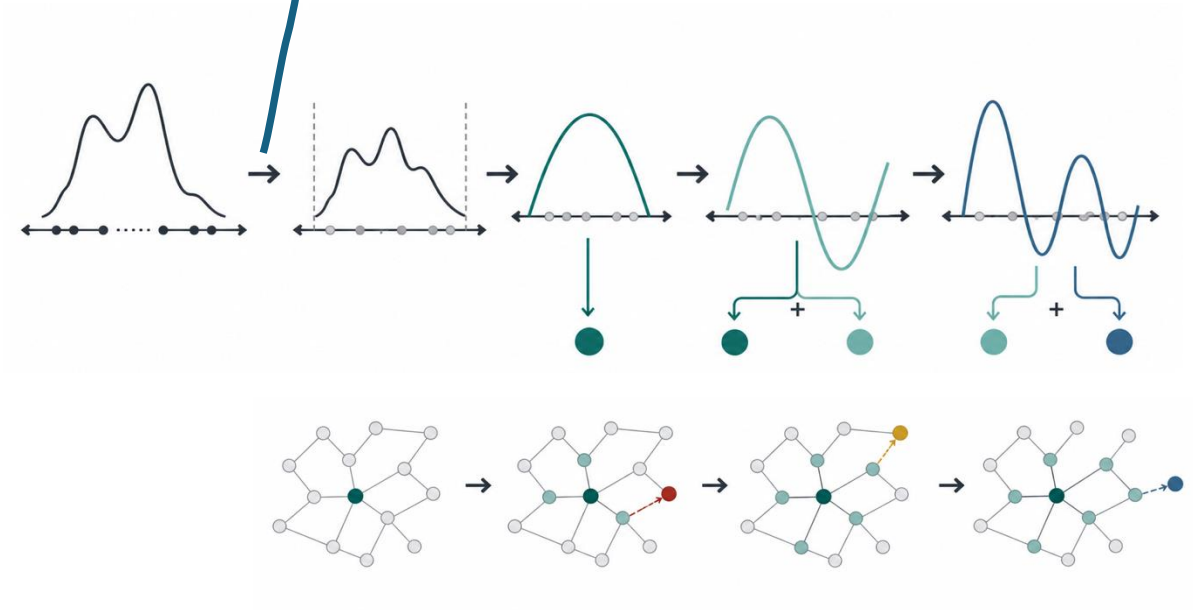
Same structure as ridge regression: $h = (\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{\Psi} + \mu\boldsymbol{R})^{-1}\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{W}g$ — a single small $(K \times K)$ linear solve gives the taps; μ trades fit accuracy for robustness.

Solve taps once, apply the filter repeatedly with local powers of the shift.

Chebyshev recurrences are local

$$\tilde{L} = (2/\lambda_{\max})L - I, \qquad H(L)x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

Rescale the Laplacian spectrum to $[-1,1]$ before evaluating Chebyshev polynomials.



CHEBYSHEV POLYNOMIALS, DEFINED BY RECURRENCE

$$T_0(s) = 1, \qquad T_1(s) = s, \qquad T_k(s) = 2sT_{k-1}(s) - T_{k-2}(s)$$
$$\Rightarrow \quad u_k = T_k(\tilde{L})x$$

THREE-TERM RECURRENCE

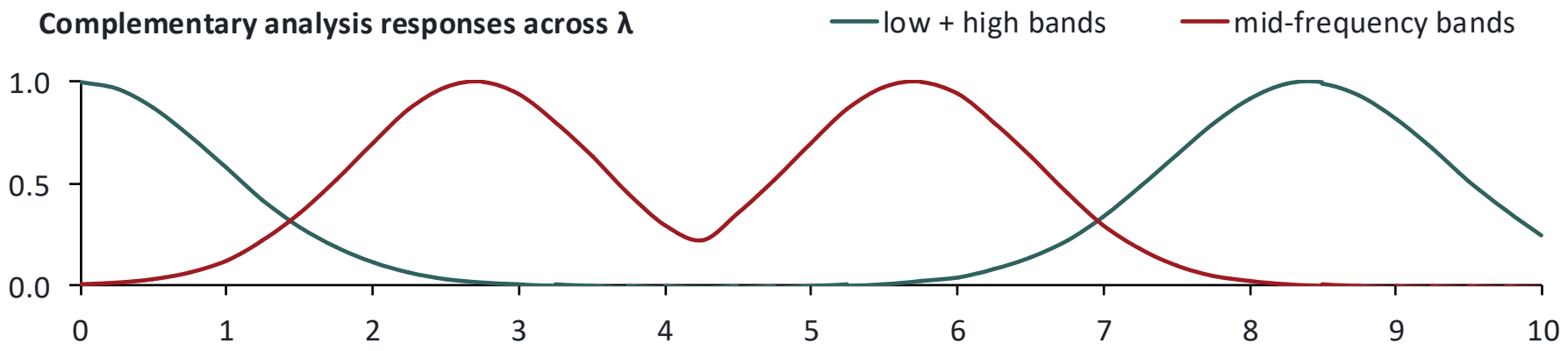
$$u_0 = x, u_1 = \tilde{L}x, u_k = 2\tilde{L}u_{k-1} - u_{k-2}$$

No eigenvectors are required
Only neighbor exchanges and feature accumulation.

Chebyshev filters are stable to evaluate and naturally distributed.

A graph filter bank tiles the spectrum into features

- A bank contains F responses $\{\hat{h}^{f}(\lambda)\}_{f=1}^{F}$
  - Each output $\boldsymbol{z}^{f} = h^{f}(\boldsymbol{S})\boldsymbol{x}$ is a graph signal
  - Low-, mid-, and high-frequency bands capture complementary structure
- Stack outputs as $\boldsymbol{Z} = [\boldsymbol{z}^{1}, \ldots, \boldsymbol{z}^{F}]$, with shape $n \times F$



Columns of *Z* are graph signals. Rows are feature vectors at nodes.

Frame bounds keep a filter bank informative

- Let $\boldsymbol{z}^{\mathrm{f}} = h^{\mathrm{f}}(\boldsymbol{S})\boldsymbol{x}$ be the outputs of F analysis filters.

$$m\|\boldsymbol{x}\|^2 \leq \sum_f \left\|\boldsymbol{z}^{\mathrm{f}}\right\|^2 \leq M\|\boldsymbol{x}\|^2$$

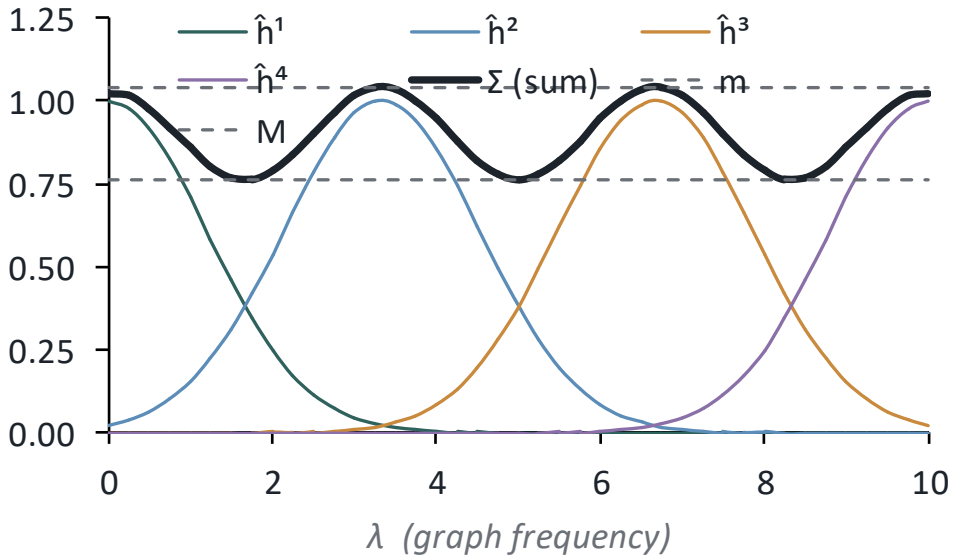
FREQUENCY-DOMAIN CONDITION

$$m \leq \sum_f \left|\hat{h}^{\mathrm{f}}(\lambda)\right|^2 \leq M$$

TIGHT FRAME (m=M)

$$\sum_{\mathrm{f}} \left|\hat{h}^{\mathrm{f}}(\lambda)\right|^2 = 1 \Rightarrow \sum_f \left\|\boldsymbol{z}^{\mathrm{f}}\right\|^2 = \|\boldsymbol{x}\|^2$$

Summed filter energy stays within [m, M]



If $m > 0$, no band is missed; controlling M prevents uncontrolled amplification.

A tight frame with $m = M = 1$ preserves total energy.

Fixed priors, learned filters adapt to data

FIXED BANK · INTERPRETABLE

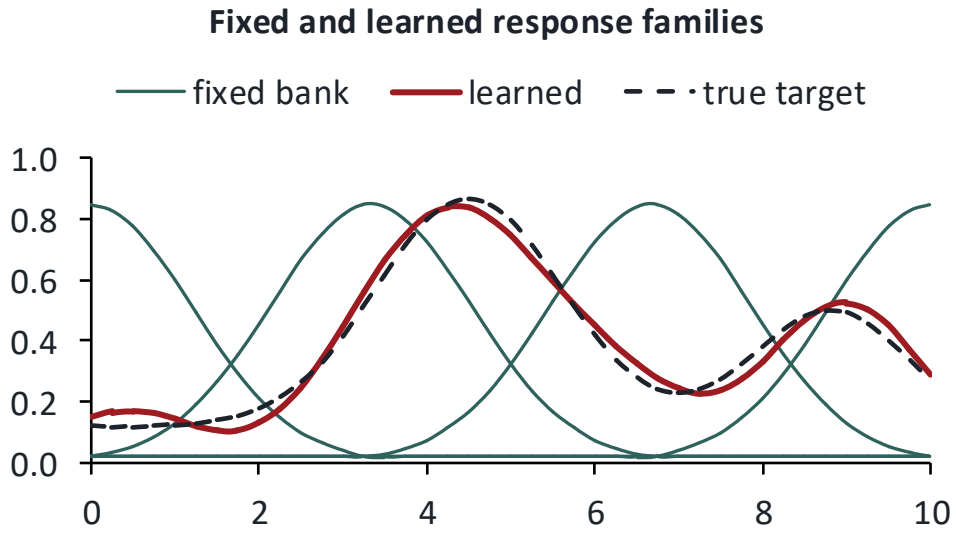
Choose heat, wavelet, or band-pass responses.
Guarantee coverage and localization.
Useful when physics defines the bands.

$\hat{h}(\lambda)$ is prescribed θ fixed

LEARNED BANK · TASK-ADAPTIVE

Optimize taps from examples.
Allocate features where the loss benefits.
Retain graph-local parameter sharing.

$\hat{h}^{\mathrm{f}}(\lambda)$ is learned θ optimized by ERM



The graph shift stays fixed; response coefficients become trainable.

Learning the taps converts a transform into a representation.

Graph learning problems differ by what is predicted

01 **GRAPH-SIGNAL TASKS**

One fixed graph supports input/output signals; example: sensor forecasting.

02 **NODE-LEVEL TASKS**

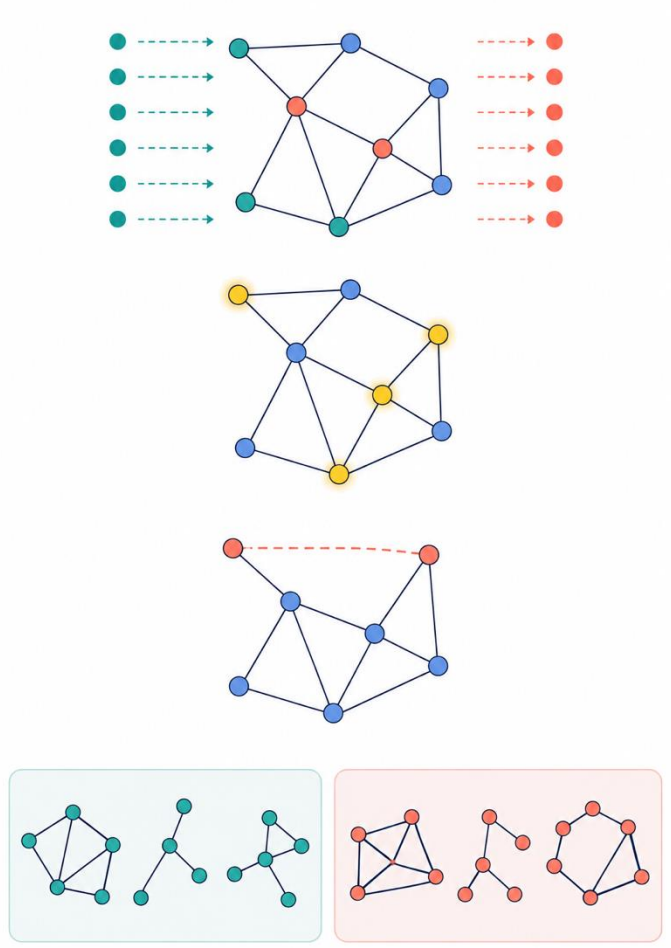
Predict labels or values for nodes from features and neighborhoods.

03 **LINK-LEVEL TASKS**

Predict whether an edge exists or should be added between two nodes.

04 **GRAPH-LEVEL TASKS**

Each graph is one sample. Predict a property for the whole graph.



PART III

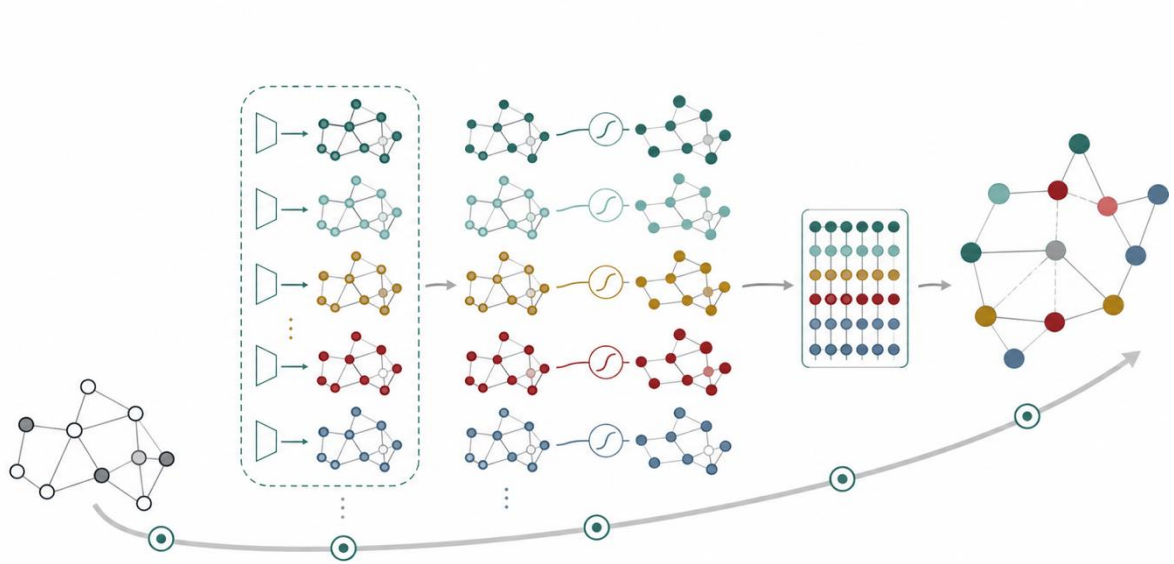
Filter banks become learned representations.

01 Add pointwise nonlinearities after graph filters

02 Cascade perceptrons to build depth

03 Carry several features at every node

04 Learn MIMO filter banks end to end



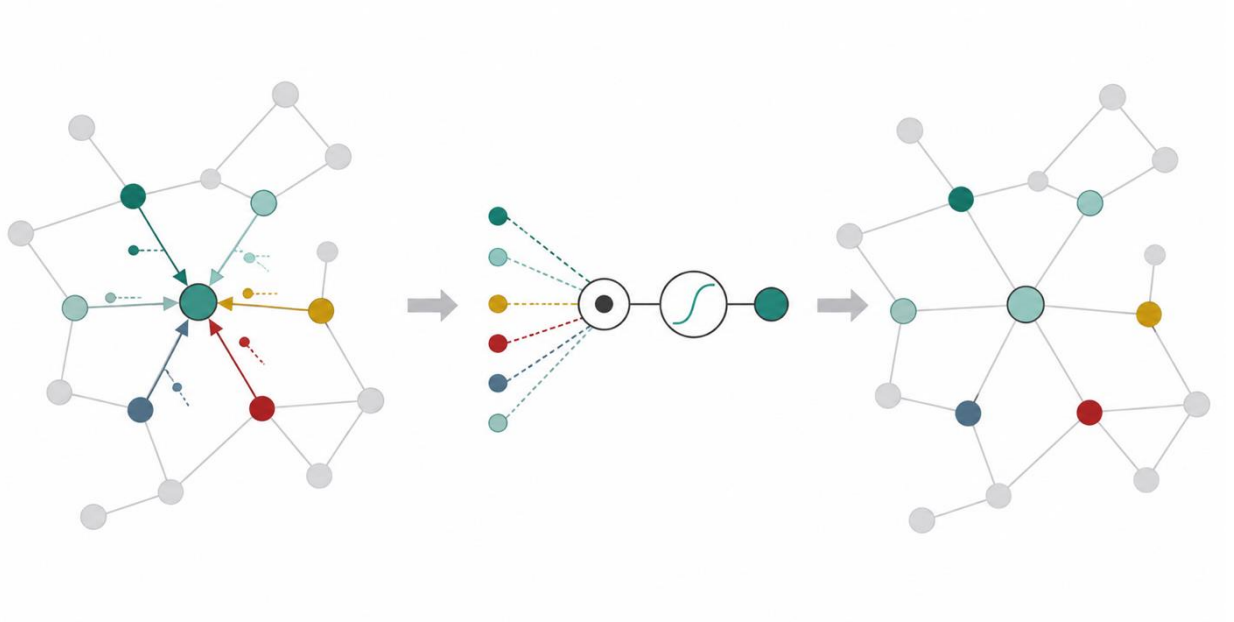
A GNN layer is a learned filter bank followed by a nodewise activation.

A graph perceptron is a filter followed by σ

First aggregate with a polynomial graph filter, then activate every node independently:

$$u = \sum_{k=0}^{K-1} h_k S^k x \,, y = \sigma(u)$$

- The filter mixes neighboring values and controls the spectral response.



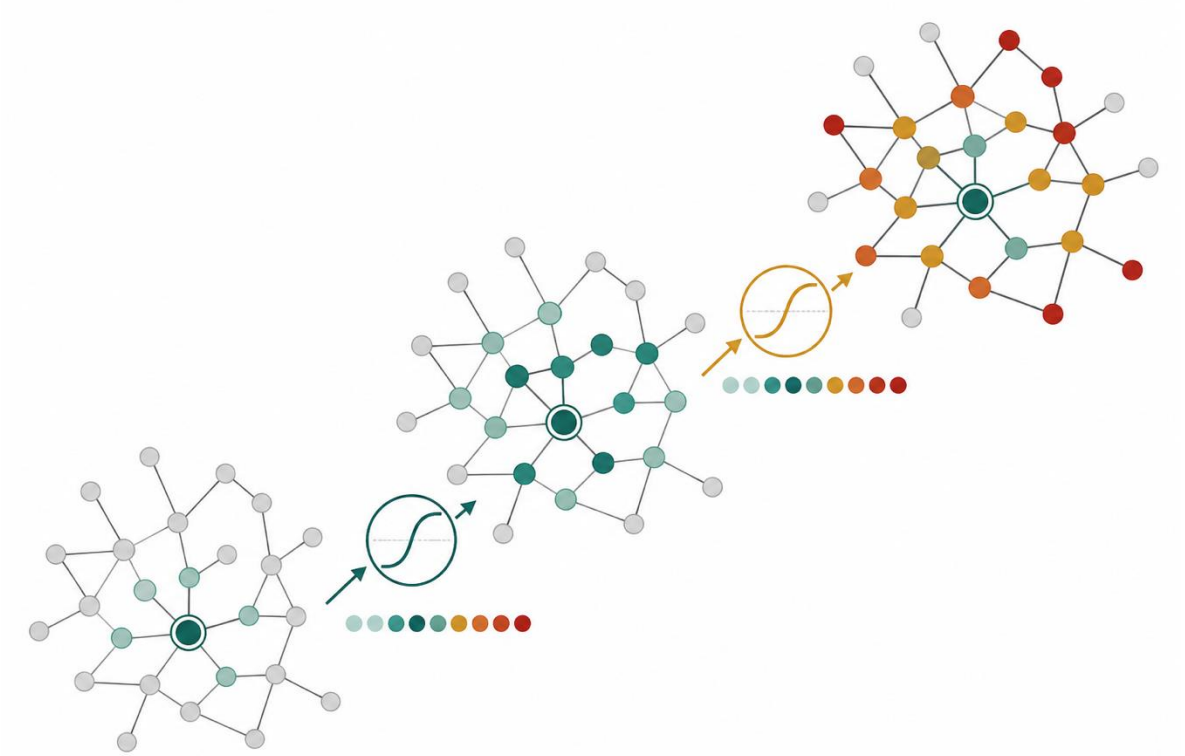
Filter locally, activate nodewise.

Cascading graph perceptrons builds depth

$$x[\ell] = \sigma(H[\ell](S)x[\ell-1]), \qquad x[0] = x$$

$$y = x[L]$$

- Each layer reuses the graph geometry but learns new taps.
- Repeating local mixing and activation expands the receptive field and composes nonlinear representations.



Depth is repeated graph filtering plus nodewise nonlinearity.

Node features turn graph signals into matrices

SCALAR SIGNAL

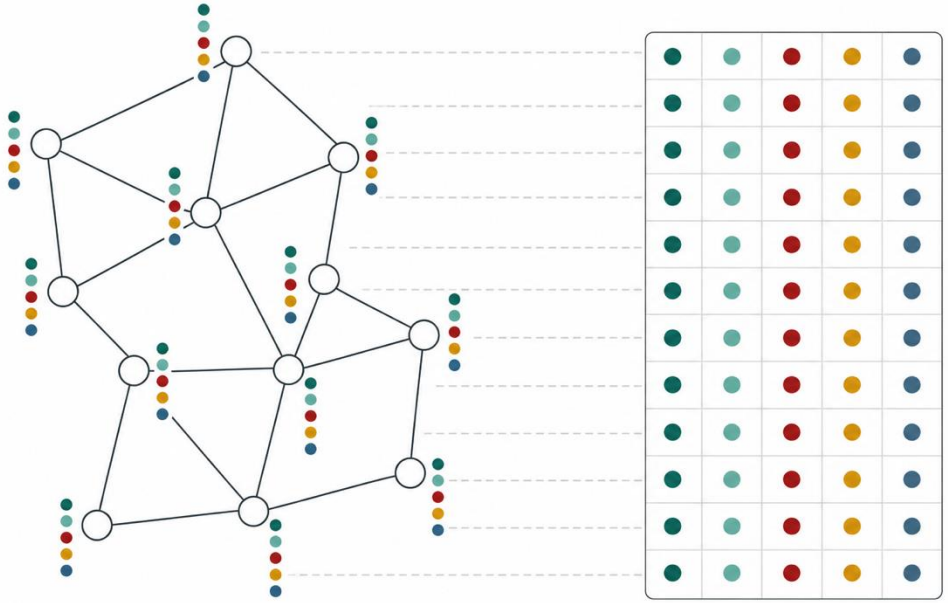
$\boldsymbol{x}$ has shape $n \times 1$

One number per node.
One response channel.
Limited representation capacity.

MATRIX SIGNAL

$\boldsymbol{X}$ has shape $n \times F$

F features per node.
Columns are graph signals.
Rows are node feature vectors.

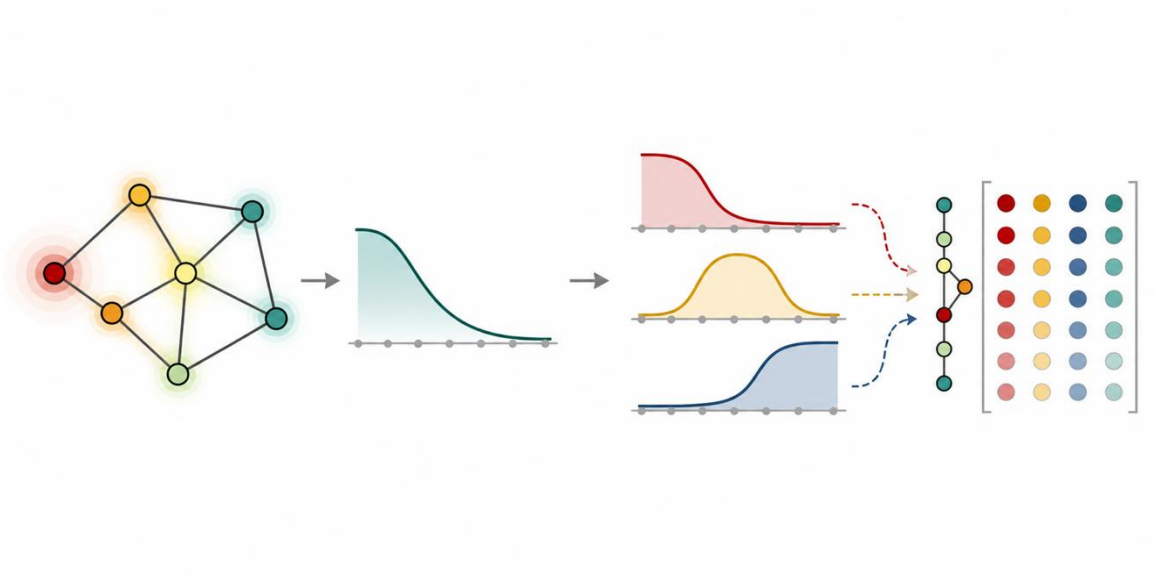


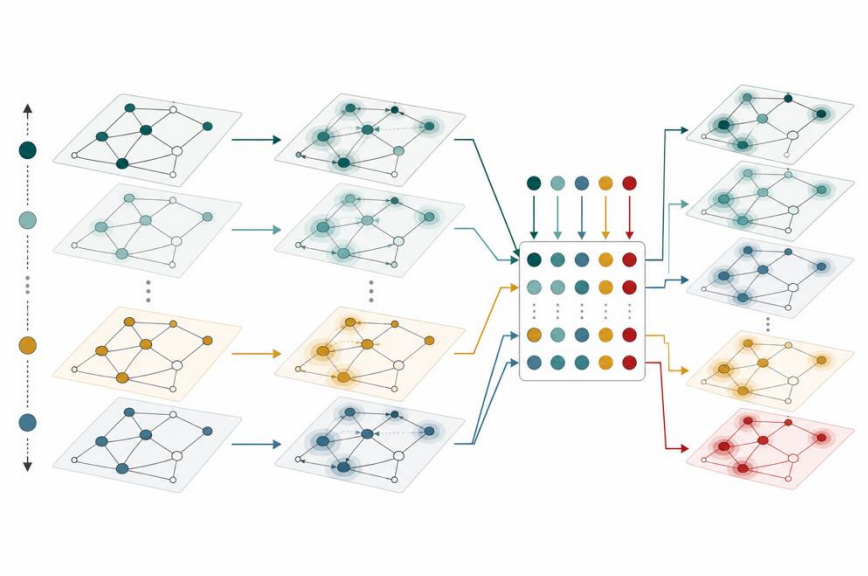
Multiple features let each node carry a learned representation.

MIMO filters mix nodes and feature channels

Let $\boldsymbol{X}$ have $n \times F_{\text{in}}$entries. At hop k, $\boldsymbol{H}_{\text{k}}$ maps F_{in} input features to F_{out} outputs.

$$\boldsymbol{Z} = \sum_{\text{k}=0}^{\text{K}-1} \boldsymbol{S}^{\text{k}} \boldsymbol{X} \boldsymbol{H}_{\text{k}}$$

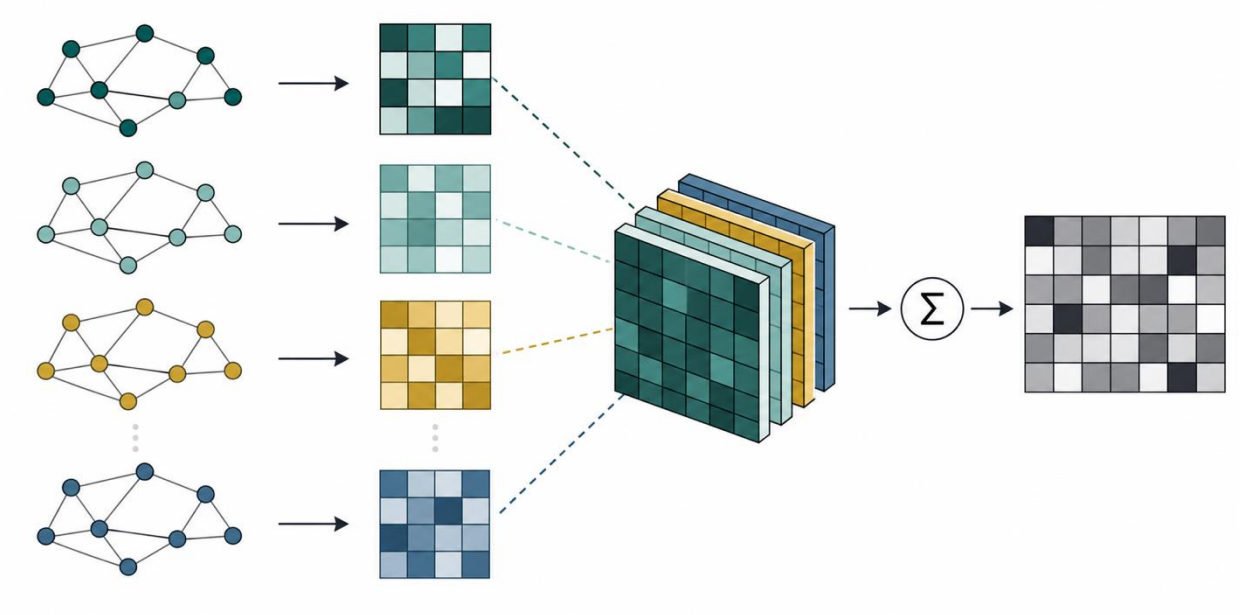




Matrix taps make the filter-bank notation compact

Scalar coefficients $h_{\mathrm{k}}^{\mathrm{fg}}$ are the entries of H_{k}.

$$\boldsymbol{Z} = \sum_{\mathrm{k}=0}^{\mathrm{K}-1} \boldsymbol{S}^{\mathrm{k}} \boldsymbol{X} \boldsymbol{H}_{\mathrm{k}}$$

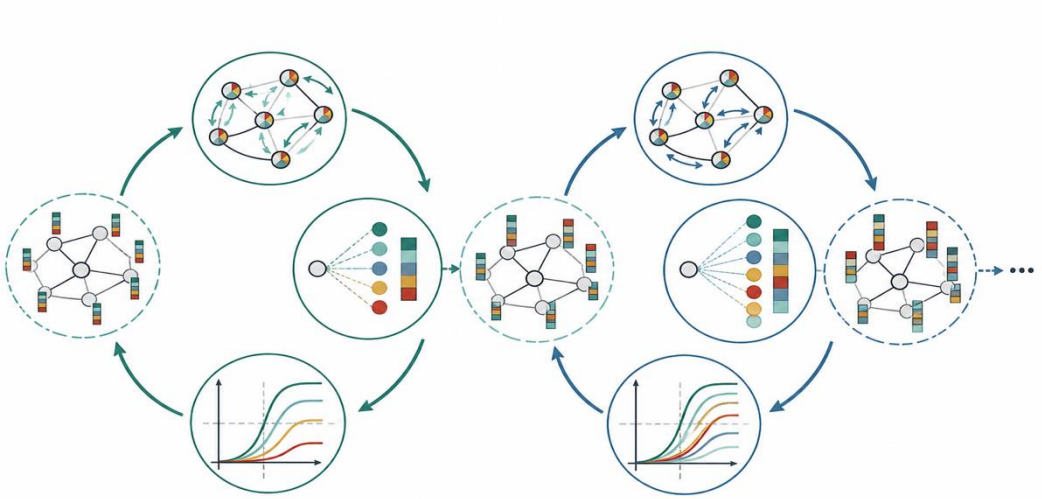


The shift mixes nodes, learned matrices mix features.

GNN layers repeat MIMO filtering and activation

$$\boldsymbol{Z}[\ell] = \sum_{\mathrm{k}=0}^{\mathrm{K}-1} \boldsymbol{S}^{\mathrm{k}} \boldsymbol{X}[\ell-1] \boldsymbol{H}[\ell, k]$$

Layer ℓ receives $F[\ell-1]$ features per node and produces $F[\ell]$ new features.



A practical GNN is a cascade of learned MIMO graph perceptrons.

ChebNet connects polynomial filters to GNNs

01 **What does ChebNet learn?**

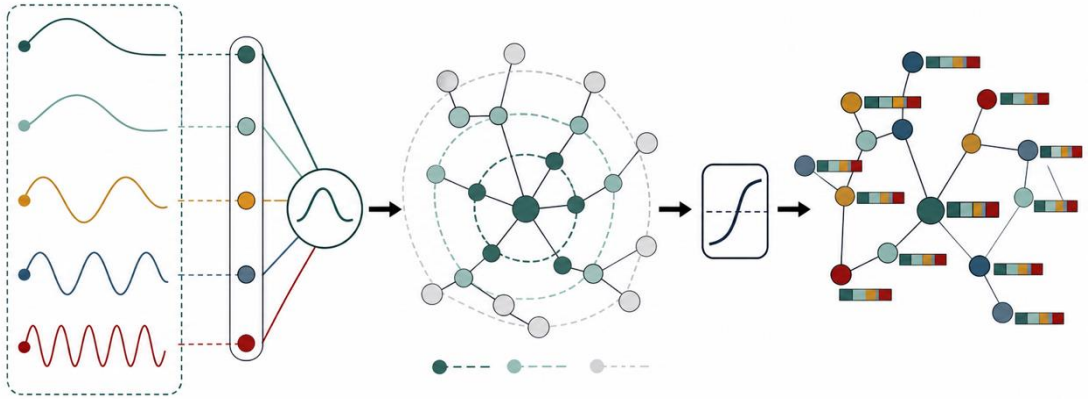
Chebyshev coefficients for each input-output feature pair.

02 **Why is it localized?**

A degree-(K-1) filter depends on at most K-1 hops.

03 **What makes it a neural layer?**

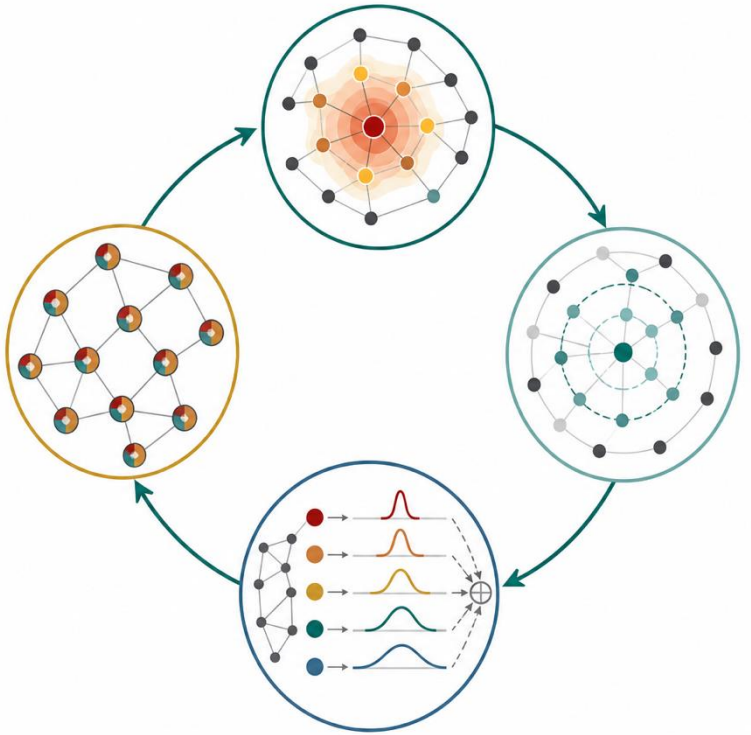
A learned filter bank is followed by a pointwise nonlinearity.



Checkpoint: identify node mixing S^{k}, feature mixing H_{k}, and activation σ in the layer equation.

TAKEAWAYS

Diffuse locally.
Learn feature banks.



01 Heat kernels model diffusion as low-pass filtering

02 Polynomial filters trade fit for communication

03 Filter banks create complementary features

04 MIMO perceptrons stack into GNNs

ChebNet learns localized spectral filters inside a nonlinear feature pipeline.