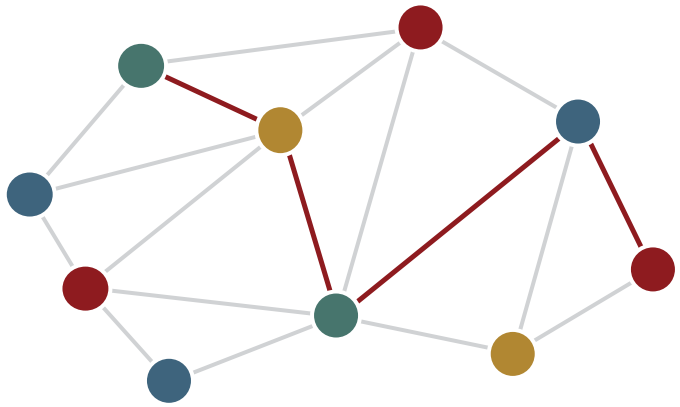
Statistical learning Over Graphs with GNNs



September 10

Zhiyang Wang · Electrical & Systems Engineering · WashU

Class 6 · statistical learning and graph machine learning tasks

Three questions organize the lecture

01 **WHAT IS LEARNED?**

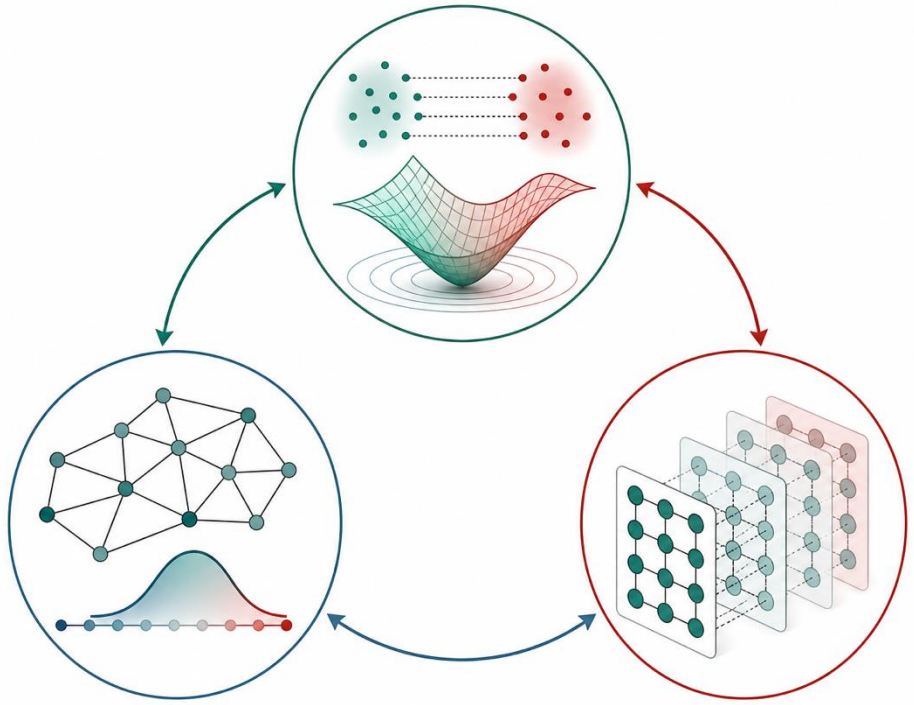
Estimate model parameters by minimizing empirical risk over training examples.

02 **WHERE DOES THE GRAPH ENTER?**

Use a shift operator to define a structured, local hypothesis class.

03 **WHICH OBJECT IS PREDICTED?**

Distinguish graph-signal, node-level, and graph-level learning problems.



Task, data, hypothesis class, and loss define the model.

Finite data stand in for an unknown population

POPULATION MODEL

(x, y) follow an unknown distribution $p(x, y)$.

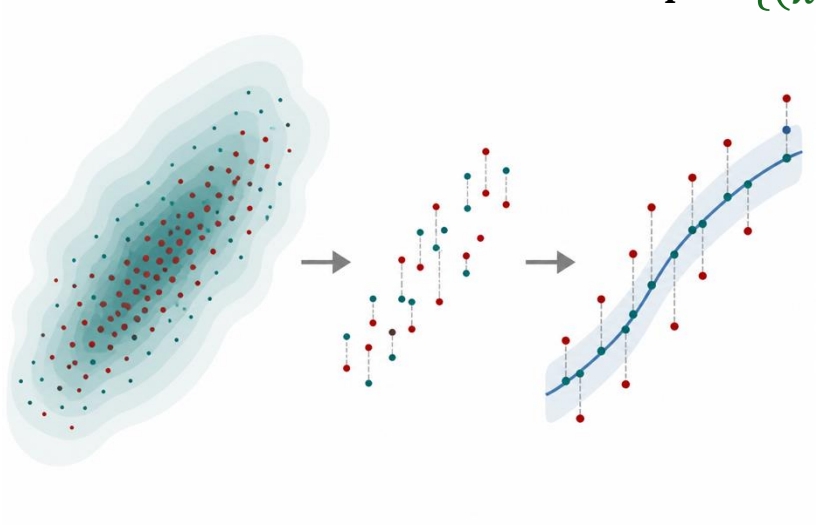
The ideal predictor minimizes expected loss.

OBSERVED TRAINING SET

Only a finite sample $T = \{(x_i, y_i)\}_{i=1}^{m}$ is observed.

Empirical averages estimate population quantities.

$$T = \{(x_i, y_i)\}_{i=1}^{m} \sim i.i.d.\ p(x, y)$$



Goal: perform well on the next draw — generalization.

Learning estimates a predictor from samples.

Empirical risk approximates population risk

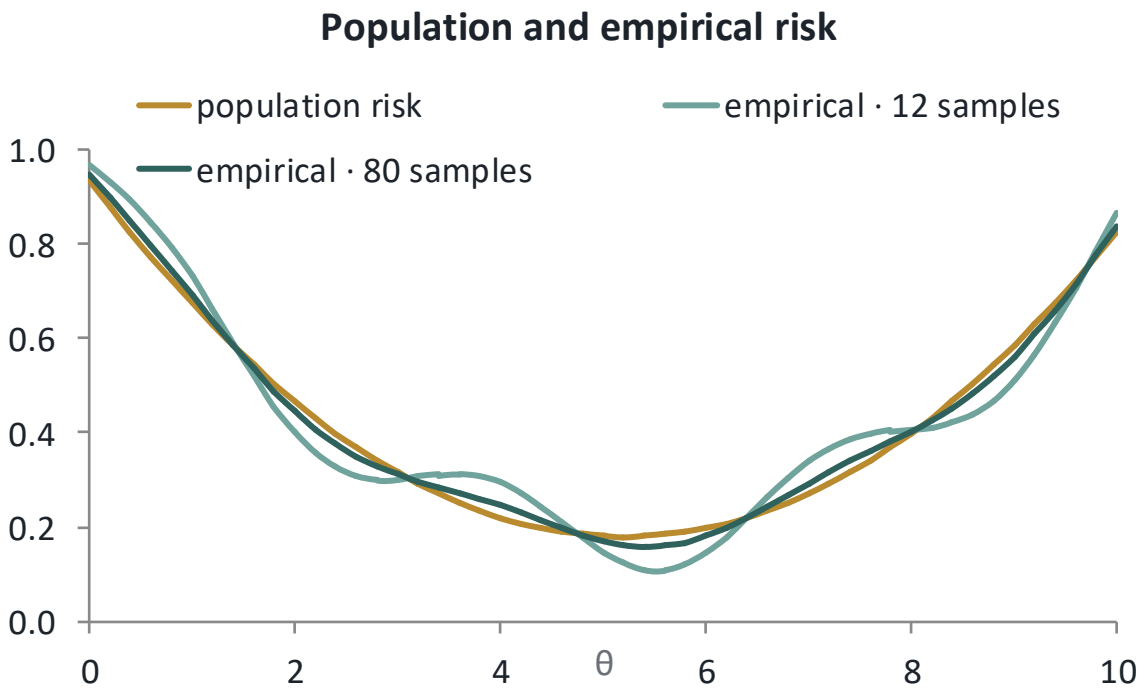
POPULATION OBJECTIVE

$$R(f) = \mathbb{E}_{p(x,y)}[\ell(y, f(x))]$$

Minimize the loss averaged over the data-generating distribution.

EMPIRICAL OBJECTIVE

$$\hat{R}_T(f) = \frac{1}{|T|} \sum_{(x_i,y_i)\in T} \ell(y_i, f(x_i))$$



Law of large numbers: for each fixed f, $\hat{R}_T(f) \to R(f)$ as the sample grows.

ERM replaces an inaccessible expectation with a computable sample average.

The Empirical Risk Minimization Principle

THE BENCHMARK · BEST POSSIBLE

$$f^* = \operatorname{argmin}_f R(f)$$

over all predictors — unattainable: p is unknown.

BEST IN THE CLASS

$$f_{\mathcal{F}}^* = \operatorname{argmin}_{f \in \mathcal{F}} R(f)$$

over function class — unattainable: p is unknown.

THE PRINCIPLE · BEST IN THE CLASS OVER SAMPLES

$$\hat{f}_T = \operatorname{argmin}_{f \in \mathcal{F}} \hat{R}_T(f)$$

computable: restrict to $\mathcal{F}$, minimize the average.

THE GAP · EXCESS RISK

$$R(\hat{f}_T) - R(f^*) = \left[R(\hat{f}_T) - R(f_{\mathcal{F}}^*)\right] + \left[R(f_{\mathcal{F}}^*) - R(f^*)\right]$$

estimation · finite-sample error *approximation · the class's bias*

- Larger training set decreases the first term, richer $\mathcal{F}$ shrinks the second term while inflates the first.
- This course's move: shrink $\mathcal{F}$ by **structure** — the graph — not merely by size.

ERM needs three ingredients

01 **TRAINING SET**

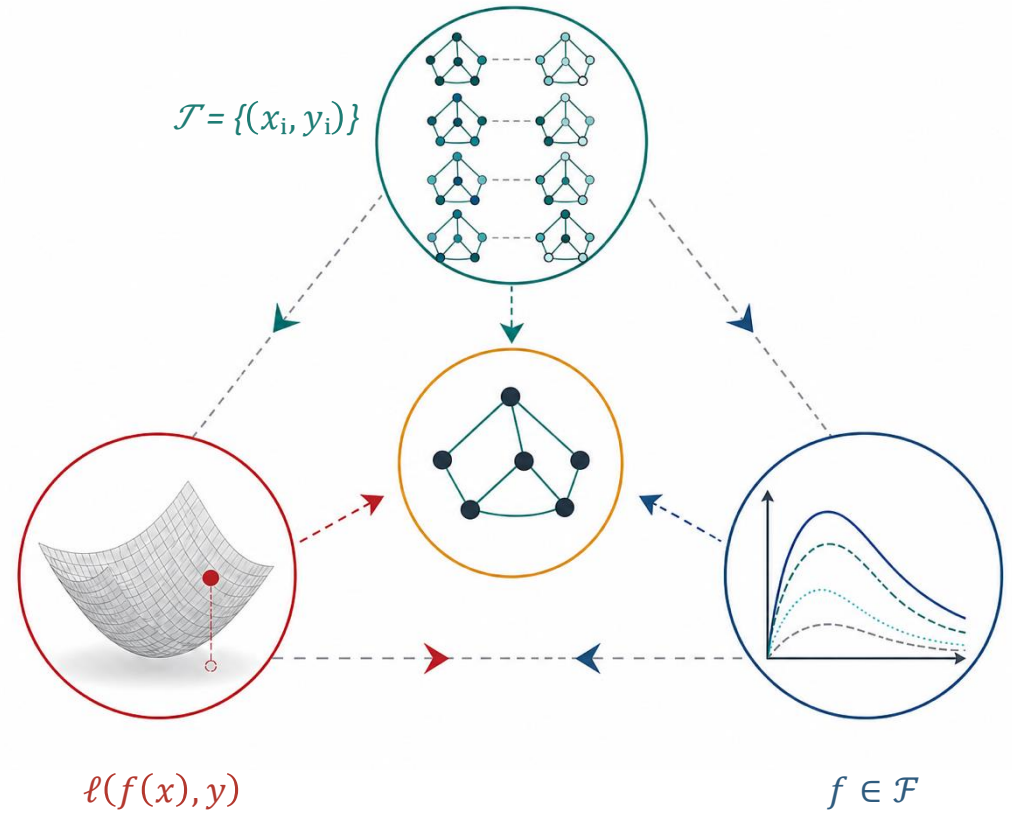
Observation pairs define what evidence the optimizer can use.

02 **LOSS FUNCTION**

Squared error, cross-entropy, or another cost scores predictions to train optimizer.

03 **HYPOTHESIS CLASS**

The allowed predictors encode the assumptions and degrees of freedom.



Hypothesis classes encode inductive bias

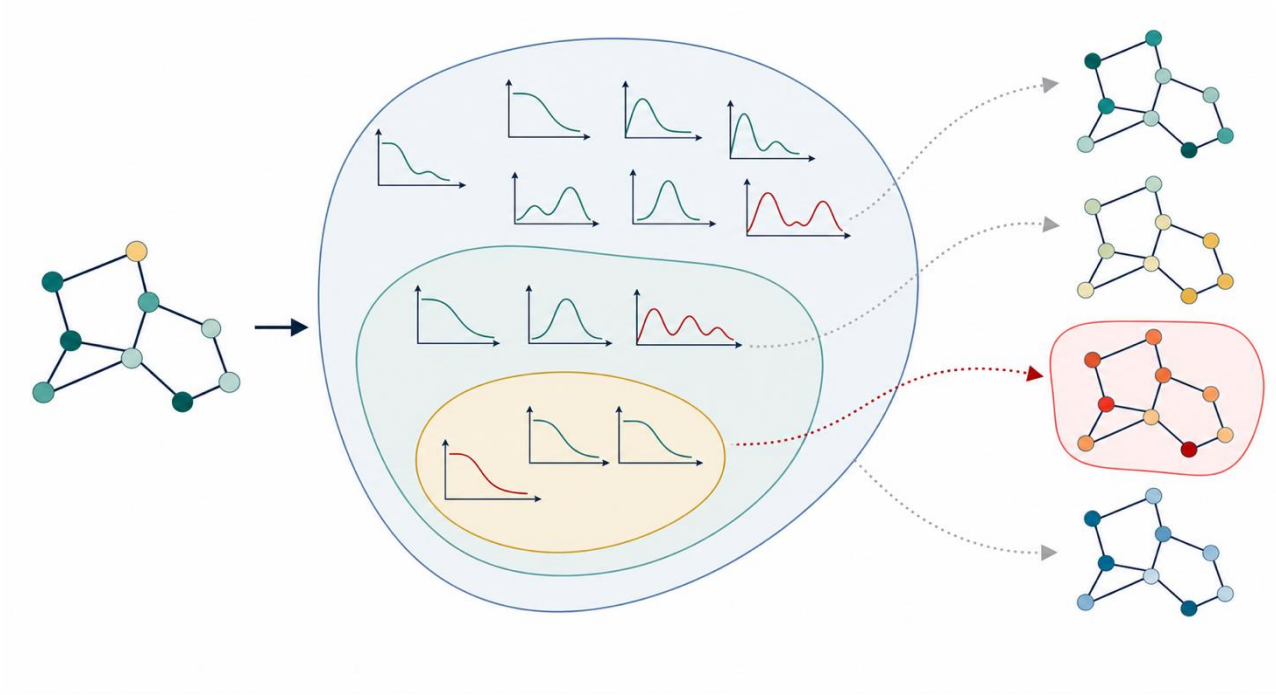
$$\mathcal{F} = \{f_\theta : \theta \in \Theta\}$$

The data choose parameters θ;
The designer chooses which functions are available.

GRAPH-AWARE FUNCTION CLASS

$$\Phi(\boldsymbol{x}; \boldsymbol{S}, h) = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x}$$

Fixing $\boldsymbol{S}$ shares parameters across nodes and restricts mixing to graph neighborhoods.



Designing a learning system begins by choosing the right function class.

The graph shift supplies structural prior information

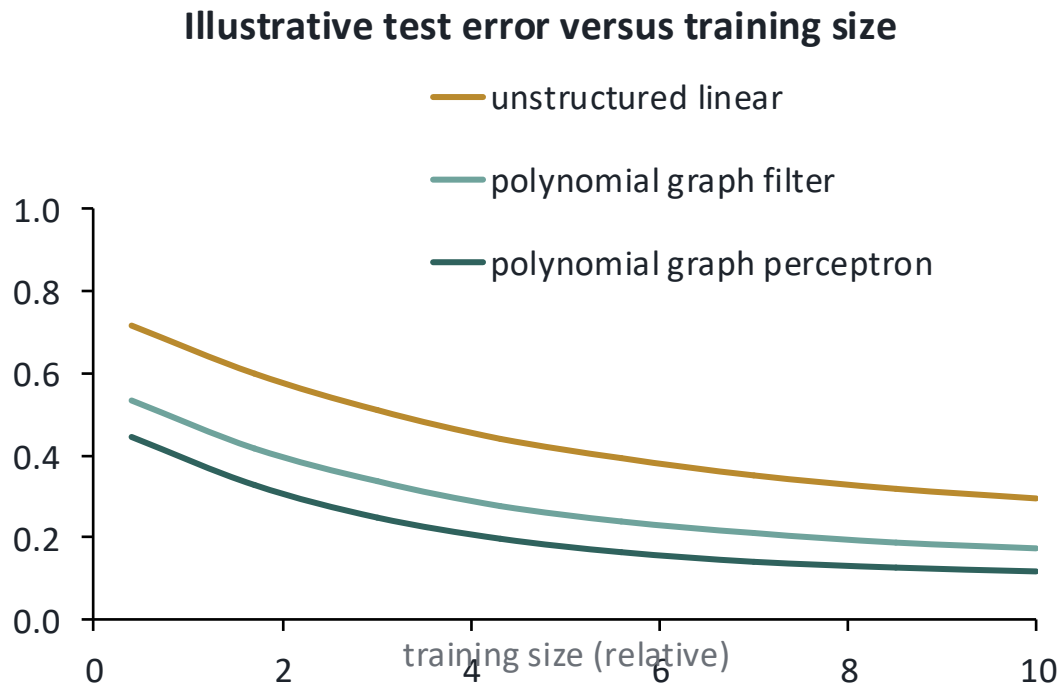
- $\boldsymbol{S}$ is fixed by the graph
- h_k is learned from examples
- The same taps are shared at every node

$$\Phi(\boldsymbol{x}; \boldsymbol{S}, \boldsymbol{h}) = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x}$$

INDUCTIVE BIAS

locality + parameter sharing + equivariance

The prior narrows the hypothesis class before any optimization begins.



The graph is given structure, learning selects how strongly to use it.

Polynomial filters form a local hypothesis class

- K learned coefficients
- $K-1$-hop receptive field
- One sparse shift multiplication per order

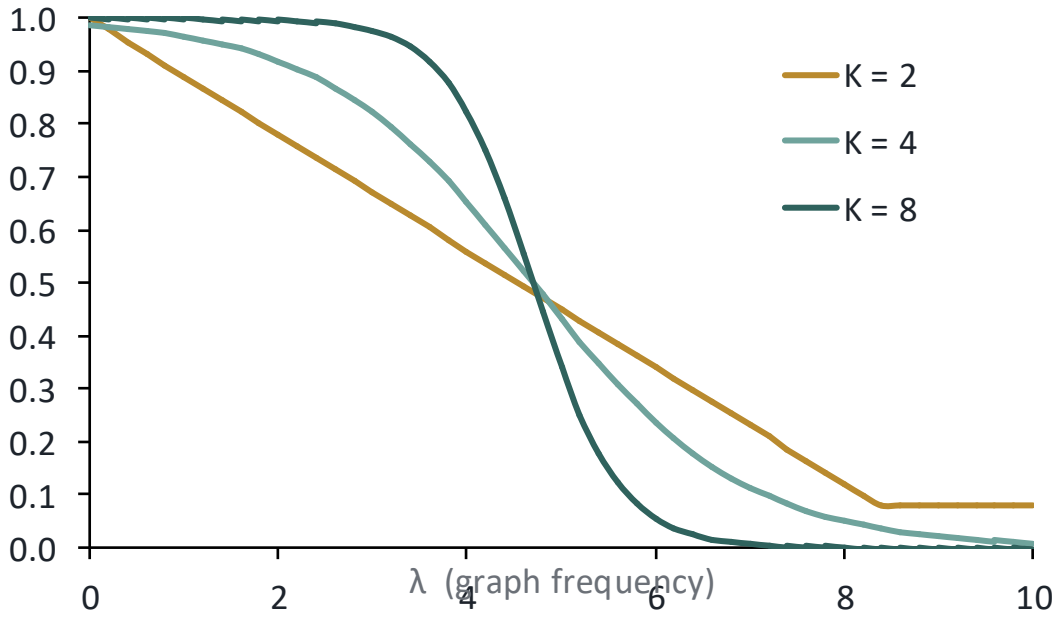
$$H(\boldsymbol{S};\mathbf{h}) = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k$$

RESTRICTED ERM

$$\boldsymbol{h}^* = argmin_{\boldsymbol{h}} \sum_{(x,y)\in T} \ell(\boldsymbol{y}, H(\mathbf{S};\boldsymbol{h})\boldsymbol{x})$$

Optimization changes the taps, while topology and locality remain fixed.

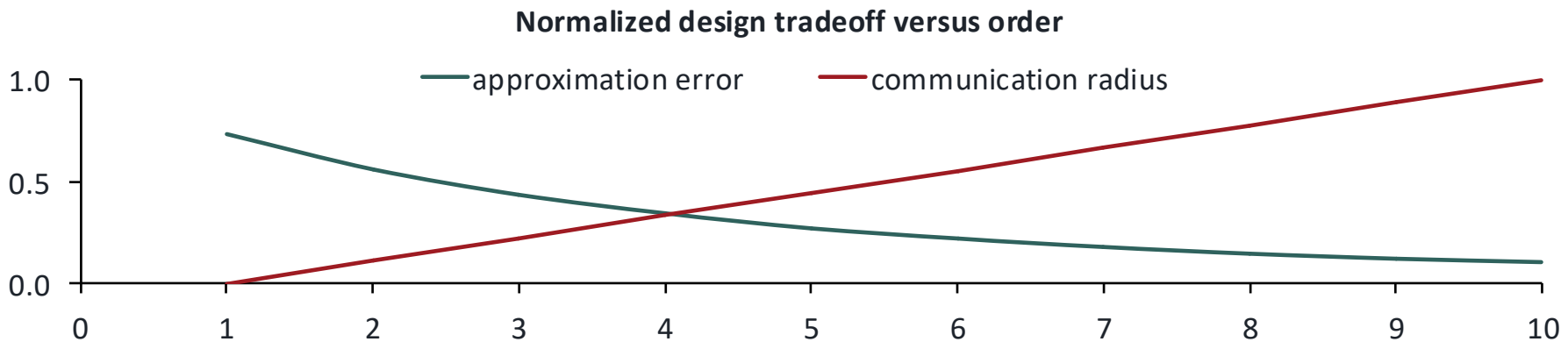
Increasing order expands the response family



A polynomial graph filter is both a signal-processing operator and a learning model.

Order K sets the capacity of the learning problem

- Small K: few taps and short reach — the class underfits
- Large K: richer hypothesis class, more parameters to estimate
- Degree $K-1$ couples capacity to a $K-1$ hop receptive field



Choosing K is choosing the bias–variance point of the GNN.

ERM learns polynomial taps from graph-signal pairs

For a fixed graph shift $\boldsymbol{S}$, each training pair supplies a prediction error:

$$\hat{\boldsymbol{y}} = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x}, \qquad \ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

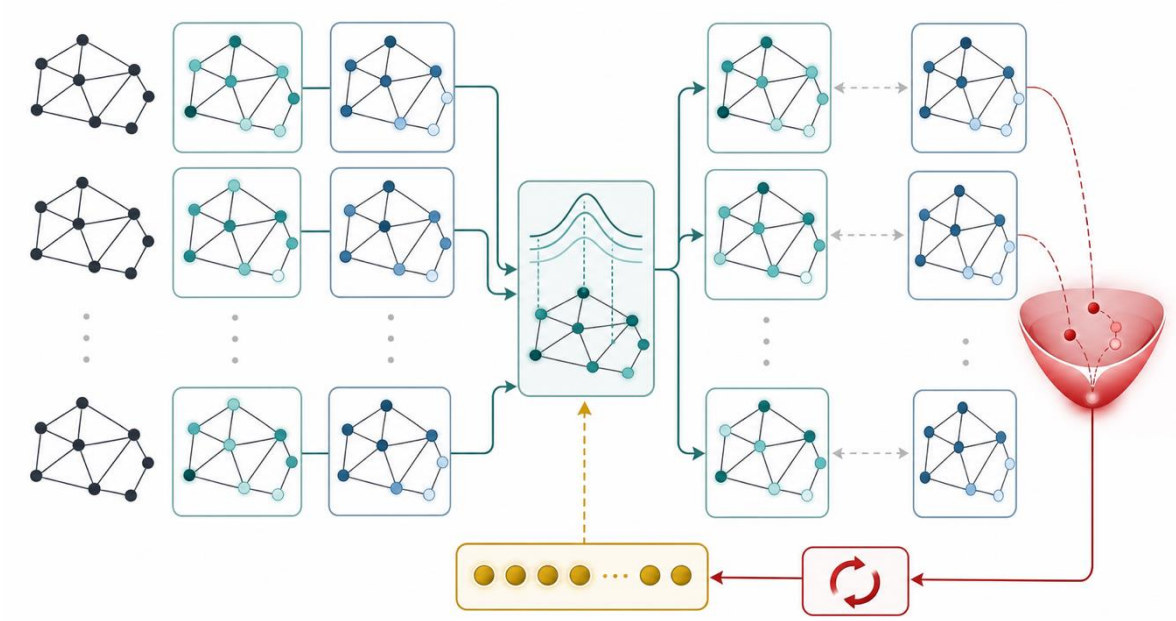
EMPIRICAL RISK · AVERAGE LOSS OVER THE TRAINING SET

$$\hat{R}_T(\boldsymbol{h}) = \frac{1}{|T|} \sum_i \ell(y_i, \hat{y}_i)$$

LOSS EXAMPLES

Squared error $\ell(y, \hat{y}) = \|y - \hat{y}\|^2$ — regression, denoising

Cross-entropy $\ell(y, \hat{y}) = -\sum_i \sum_c y_{i,c} \log \hat{y}_{i,c}$ — node classification



- Gradient-based optimization aggregates those errors across the training set:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} - \eta \nabla_{\boldsymbol{h}} \hat{R}_T(\boldsymbol{h})$$

The graph remains fixed, data determine the spectral response through $\boldsymbol{h}$.

Designed and learned filters use the same basis

DESIGNED COEFFICIENTS

Specify $g(\lambda)$; fit θ by approximation.

Interpret spectral bands before seeing data.

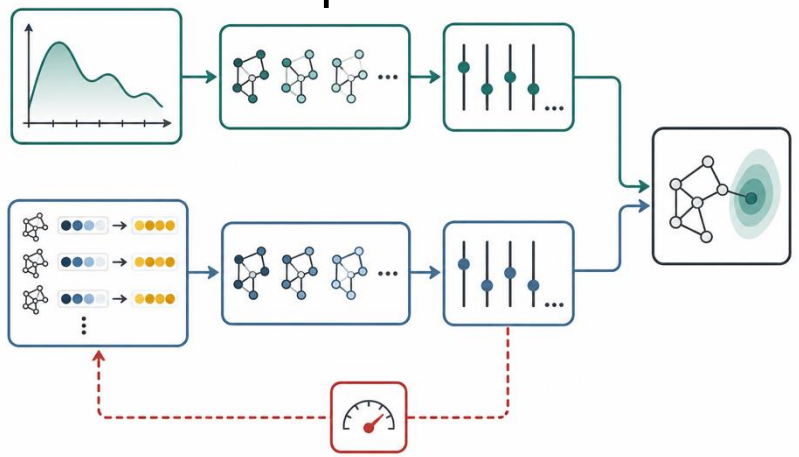
$$h_{design} = \operatorname*{argmin}_{h} \sum_{i} \left| g(\lambda_i) - \sum_{k=0}^{K-1} h_k \lambda_i^k \right|^2 ,$$

LEARNED COEFFICIENTS

Use labeled examples; optimize θ by ERM.

Adapt the response to the task loss.

$$h_{learn} = \operatorname*{argmin}_{h} \frac{1}{|T|} \sum_{(x,y)\in T} \ell\left(y, \sum_{k=0}^{K-1} h_k S^k x\right).$$



The polynomial operator is unchanged; only the rule for choosing coefficients changes.

Stack it: the multi-layer graph perceptron

$$x_\ell = \sigma\left(\sum_{k=0}^{K-1} h_{\ell k}\mathbf{S}^k x_{\ell-1}\right), \qquad \ell = 1, \ldots, L, \quad \boldsymbol{x_0} = \boldsymbol{x}$$

- Each layer is exactly the perceptron: one graph filter, then a pointwise σ.
- $\boldsymbol{x}_l$ is the ℓ-layer embedding — still a graph signal — and the output is $\boldsymbol{y} = \boldsymbol{x}_l$ at $\ell = L$.
- Two architecture knobs: depth L and per-layer order K, the output can depend on nodes at most $\sum_\ell (K_\ell - 1)$

$$\boldsymbol{y} = \Phi(\boldsymbol{x}; \mathbf{S}, \mathcal{H}), \qquad \mathcal{H} = \{h_{\ell k}\}$$

One symbol for the whole stack — and ERM trains $\mathcal{H}$ exactly as it trained three taps.

Depth composes cheap local filters into an expressive nonlinear map.

Full-fledged GNNs: many features at once

- Real node data is multi-dimensional
  drones carrying 3-D positions, users carrying profiles: one feature row per node, $\boldsymbol{X} \in \mathbb{R}^{\mathrm{nxd}}$.

$$\mathbf{X}_\ell = \sigma\left(\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right), \ \mathbf{H}_{\ell k} \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$$

- $\boldsymbol{S}^{\mathrm{k}}$ multiplies on the left: **diffusion over the graph.**
- $\mathbf{H}_{\ell k}$ multiplies on the right: **mixing across features.**
- Together: a MIMO filter bank per layer — this is what "message passing" computes.

One layer, two independent mixings: the graph diffuses across nodes, the weights mix across features.

Everything above extends to GNNs

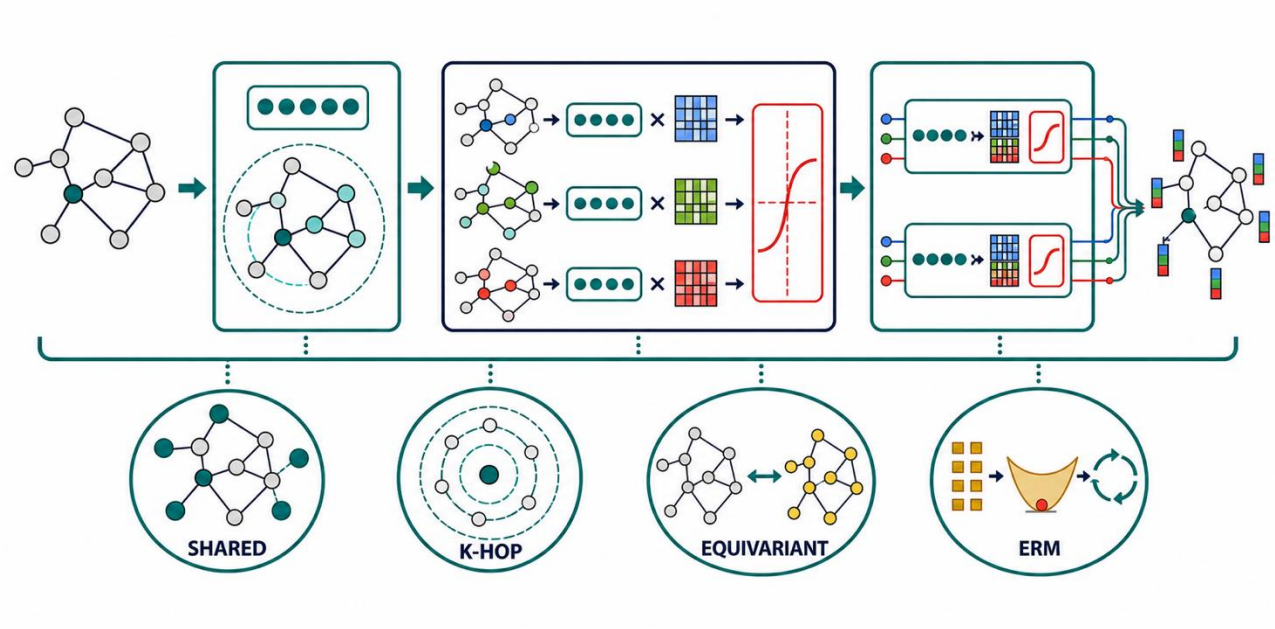
FROM GRAPH FILTERS TO GNNs

The same statistical-learning story carries over to GNNs.

$$\boldsymbol{x}^{(l+1)} = \sigma\left(\sum_k h_k \boldsymbol{S}^k \boldsymbol{x}^{(l)}\right)$$

A GNN is a filter bank plus nonlinearities — the ERM setup is unchanged.

$$\min_{\mathcal{H}} \frac{1}{|T|} \sum_i \ell\left(\Phi(x_i; \boldsymbol{S}, \boldsymbol{H}), y_i\right)$$



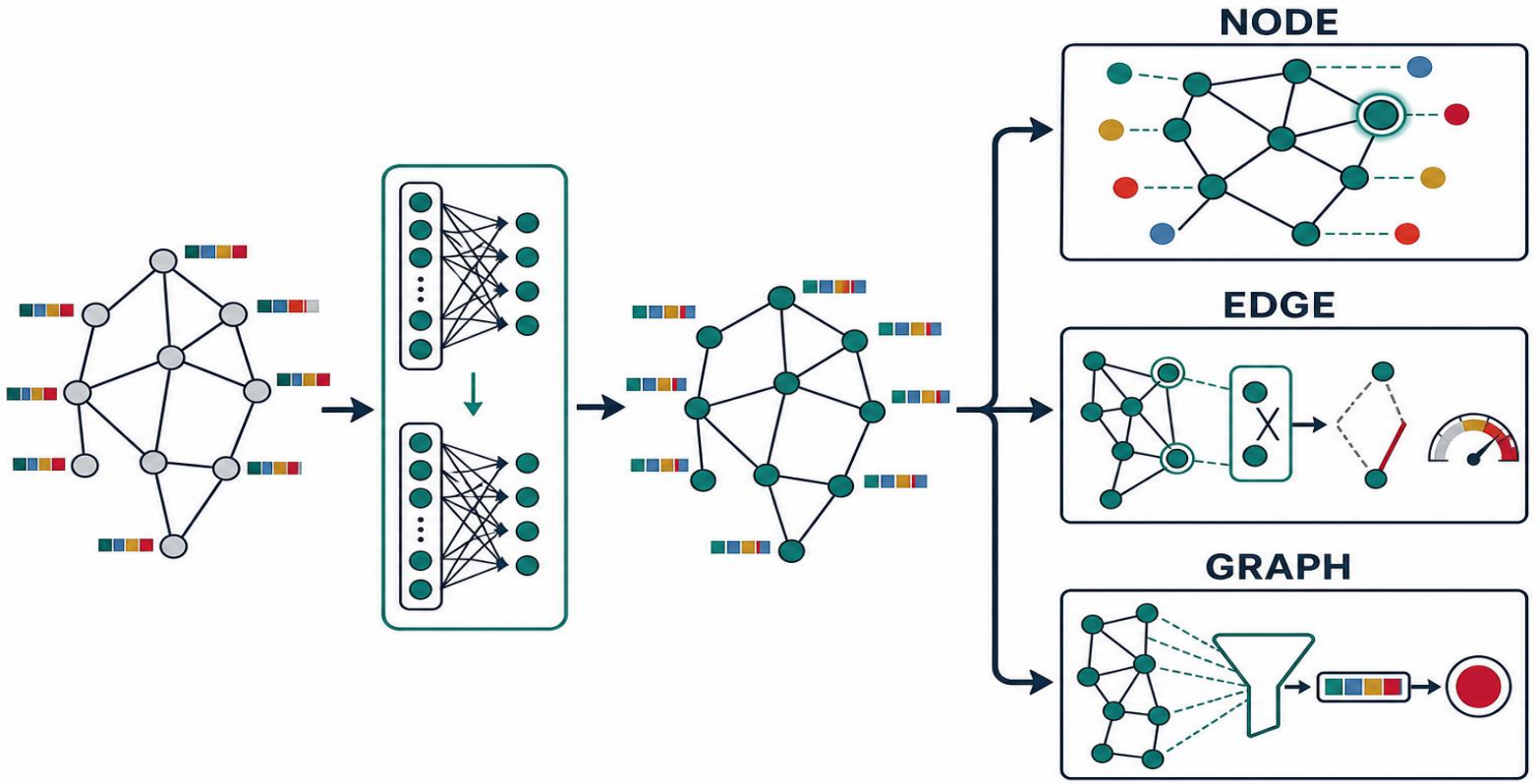
Same ERM, same graph inductive bias — **only the hypothesis class is richer**.

Nothing in the analysis was specific to linear filters — it all extends to GNNs.

Match the GNN to the prediction unit of the task

The same backbone serves different learning tasks --- the output head and loss change.

- Node tasks read out per-node values; edge tasks score node pairs; graph tasks pool every node into one embedding.



Node-level learning predicts values on nodes

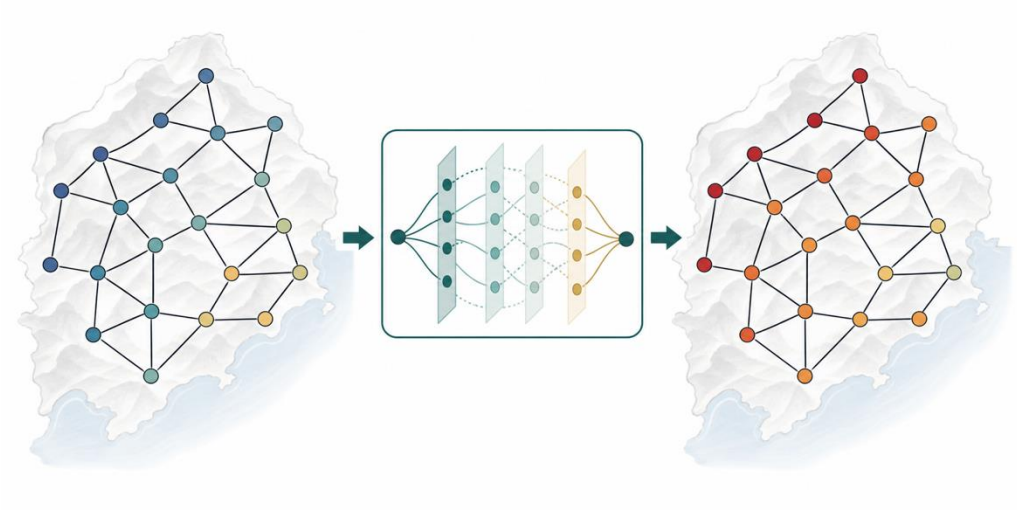
Example · weather-station forecasting

$$\hat{\boldsymbol{y}}_{future} = H(\mathbf{S}; \boldsymbol{h})\boldsymbol{x}_{past}$$

- The station graph and shift $\boldsymbol{S}$ are given.
- Each date supplies an input-output signal pair.

The loss compares predicted and recorded temperatures at all stations.

Use temporal splits to avoid leaking future conditions into training.



Samples change over time, the graph support stays fixed.

Product ratings form a signal on the user graph

INPUT + GRAPH

Nodes: users; edges: similarity weights.

For product p: $x_u^{(p)} = r_u^{(p)}$ if known; $m_u^{(p)} = 1$.

MODEL + OUTPUT

$$\hat{\boldsymbol{r}}^{(\mathrm{p})} = \Phi([\boldsymbol{x}^{(\mathrm{p})}, \boldsymbol{m}^{(\mathrm{p})}]; \boldsymbol{S}, \boldsymbol{H}), \text{output} = \hat{r}_{u^*}^{(p)}$$

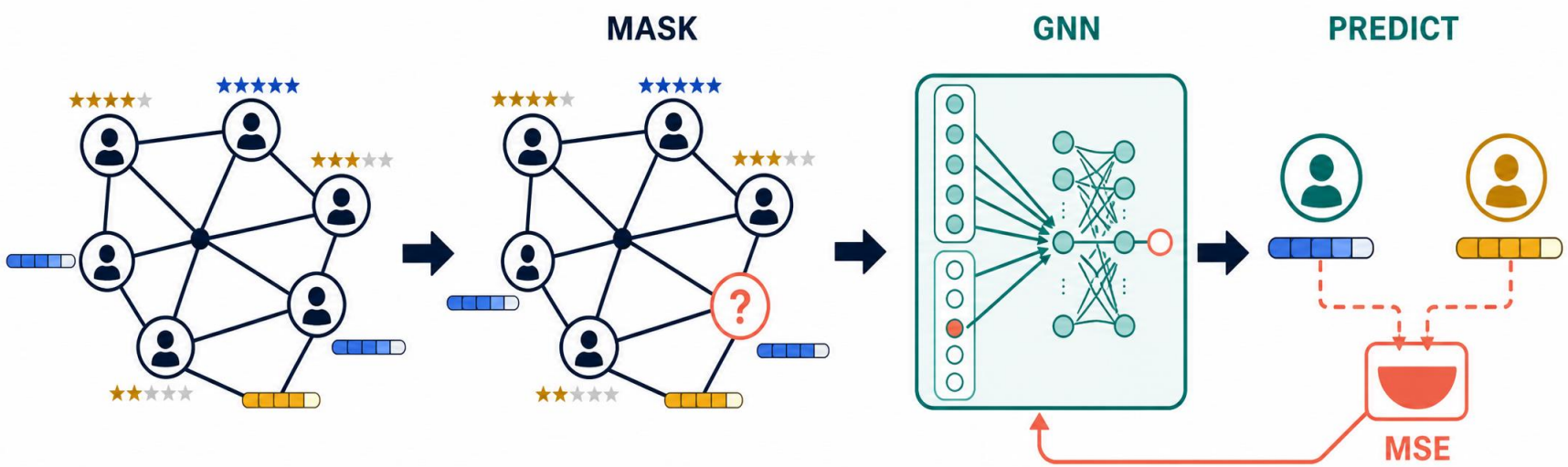
Output: $\hat{r}_{u^*}^{(p)}$ at the unrated target user.

One user node is predicted; each product supplies one rating signal.

Pointwise rating makes recommendation trainable

Training example: hide one observed $r_u^{(p)}$, then reconstruct it from the remaining users.

$$\ell(u,p) = \left[r_u^{(p)} - \Phi(\boldsymbol{x}^{(\mathrm{p})} \odot \boldsymbol{m}^{(-\mathrm{u})}, \boldsymbol{m}^{(-\mathrm{u})}; \boldsymbol{S})_{\mathrm{u}} \right]^2$$



Evaluation: split ratings first; report RMSE and MAE only on held-out entries.

User-graph rating prediction, end to end

01 **BUILD USER GRAPH —** one node per user; connect similar users from profiles or co-rated products - Pearson correlation.

02 **CREATE PRODUCT SIGNAL —** put known ratings for product p in $\boldsymbol{x}^{(\mathrm{p})}$; mark them with mask $\boldsymbol{m}^{(\mathrm{p})}$.

03 **EMBED WITH THE GNN —** two MIMO layers $\boldsymbol{Z} = \sigma(\Sigma_{\mathrm{k}} \boldsymbol{H}_k \boldsymbol{S}^{\mathrm{k}} \boldsymbol{X})$.

04 **PREDICT WITH THE GNN —** aggregate similar users on $\boldsymbol{S}$ and read out $\hat{r}_u^{(p)}$ at the target node.

05 **TRAIN AND MEASURE —** minimize masked MSE; validate on held-out ratings; report RMSE and MAE.

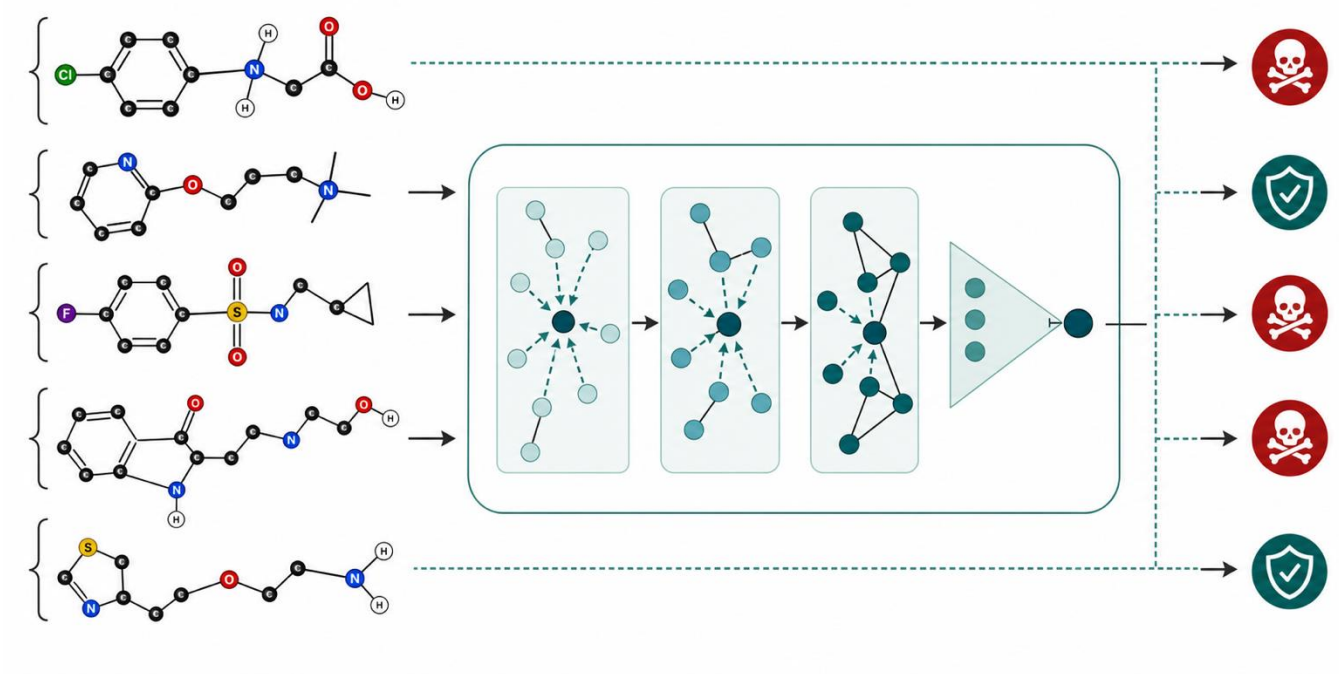
06 **SERVE —** for user u^* and unrated product p, run the product signal and return $\hat{r}_{u^*}^{(p)}$.

The task is graph-signal interpolation: predict one missing rating at one user node.

Molecular graphs map to one toxicity probability

Input $\boldsymbol{X}_G$: element, charge, aromaticity. Graph $\boldsymbol{S}_G$: typed chemical bonds.

$$\mathbf{Z}_G = \sigma\left(\sum_k \boldsymbol{H}_k \mathbf{S}_G^k \mathbf{X}_G\right),\ \mathbf{z}_G = POOL(\mathbf{Z}_G),\ p_G = \sigma(\mathbf{w}^\top \mathbf{z}_G)$$



One molecule is one sample. Global pooling makes one graph representation.

Binary cross-entropy trains molecular classification

TARGET + LOSS

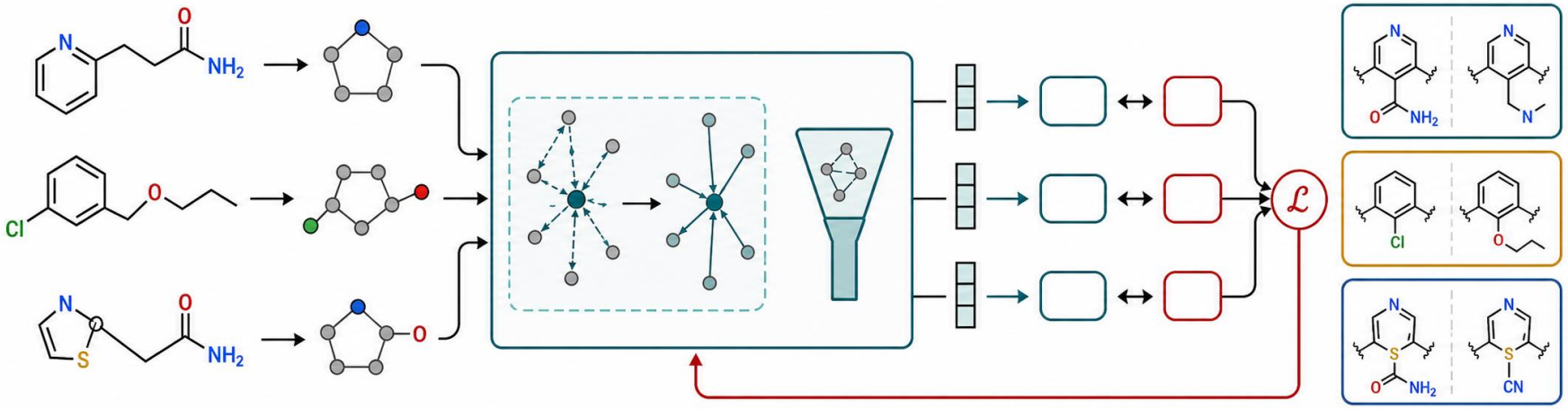
$$p = Pr(toxic \mid \mathbf{G}), \qquad y \in \{0,1\}$$

$$\ell = -\, y \log p - (1-y)\log(1-p)$$

SPLIT + METRICS

Scaffold split holds out chemical families.

Metrics: ROC-AUC and PR-AUC.



Engineering unit: one graph in, one probability out, one loss term per molecule.

Molecule classification, end to end

01 **PARSE THE DATA —** atom–bond graphs; assay outcomes → labels $y \in \{0,1\}$.

02 **FEATURIZE —** X_G: one-hot element, charge, aromaticity; S_G: bond-typed adjacency — one graph per molecule.

03 **EMBED ATOMS —** three MIMO layers produce per-atom embeddings Z_G.

04 **READ OUT —** mean-pool to a single vector z_G; linear + sigmoid gives $p_G = Pr(toxic|G)$.

05 **TRAIN AND SPLIT —** binary cross-entropy; scaffold split keeps whole chemical families out of training.

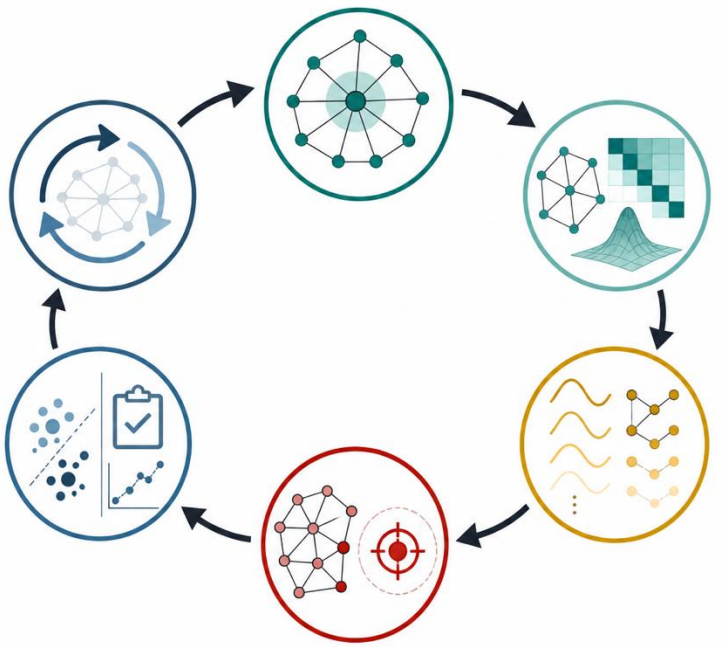
06 **DEPLOY —** rank a screening library by p_G; send the top slice to the wet lab.

The prediction unit is the molecule — pooling is where node-level becomes graph-level.

A disciplined workflow links the task to the model

design = (prediction unit, $\boldsymbol{S}$, basis, K, loss, split)

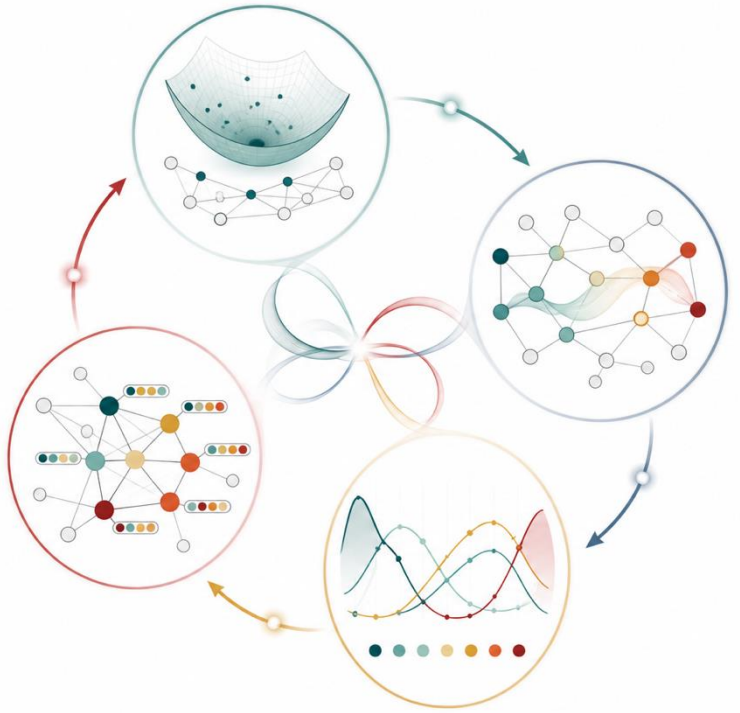
Choose the sample and output first; then select the graph prior, polynomial class, training objective, and evaluation protocol.



Architecture choices are meaningful only relative to a defined learning problem.

TAKEAWAYS

Define the task.
Then learn the filter.



01 Prediction unit sets samples and readout

02 Graph shift supplies the structural prior

03 Polynomial basis enforces local computation

04 ERM learns coefficients for the task

A GNN is a graph filter bank made trainable --
the task defines the loss, the graph defines the prior.