DEEP GRAPH COARSENING: LEARNING TO CONTRACT EDGES

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph coarsening reduces the number of nodes in a graph while preserving the structural properties that downstream tasks rely on. Classical coarsening methods are fixed algorithms designed heuristically to preserve a particular property of the graph, such as the low end of the Laplacian spectrum. They do not adapt to the graph domain at hand, and targeting a different property requires designing a new heuristic. We propose to learn the coarsening operator instead, directly from the objective it should optimize. We demonstrate that this is possible. Our general framework lets a graph neural network assign scores to the edges of a graph, and then contract a maximum-weight forest with a given number of edges. To address that the coarsening operation is discrete and non-differentiable, we use a straight-through estimator and smoothen the objective with noise. We evaluate our method using a Laplacian spectrum-preserving objective. Across nine graph datasets and three reduction ratios, the learned operator outperforms the best classical baseline in 24 of 27 settings and on all nine datasets at the highest reduction ratio (75%). Notably, these include methods explicitly designed for this objective, whereas our method is general.

1 INTRODUCTION

Graphs are the standard representation of relational data in social networks, biological systems, recommender systems, and knowledge graphs. They encode dependencies between entities that independent data points cannot capture, but exploiting this relational structure comes at a computational cost. Many graph algorithms rely on linear algebra or combinatorial operations that become increasingly expensive as the number of nodes and edges grows.

Graph coarsening reduces this cost by replacing an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a smaller graph $\widehat{\mathcal{G}}$ whose supernodes aggregate nodes of $\mathcal{G}$. Computation can then be performed on $\widehat{\mathcal{G}}$ and the results lifted back to the original graph when needed. For this to work, the smaller graph must preserve the properties that the computation relies on.

What the coarse graph needs to preserve depends on the application. In multilevel graph partitioning, coarsening is used to build smaller instances on which cut objectives can be optimized and refined (Karypis & Kumar, 1998; Dhillon et al., 2007). In electrical network reduction, Kron reduction preserves a Schur-complement representation of the network Laplacian (Dörfler & Bullo, 2013). In Laplacian-based spectral methods, such as spectral clustering, the relevant quantities are the low-frequency eigenvalues and eigenvectors of the Laplacian (von Luxburg, 2007; Loukas, 2019). More recently, graph coarsening has also been used to scale graph generation, where a model learns to invert a coarsening sequence to build large graphs iteratively through local expansions (Bergmeister et al., 2024).

Classical coarsening methods, such as heavy edge matching, algebraic distance, affinity, Kron reduction, and local variation (Karypis & Kumar, 1998; Ron et al., 2011; Livne & Brandt, 2012; Dörfler & Bullo, 2013; Loukas, 2019), use fixed rules designed to preserve particular graph properties. Targeting a different objective requires choosing or designing another heuristic. Moreover, the optimal coarsening strategy can depend on the graph distribution. Graphs from a particular domain, such as molecular, biological, social, citation, or product graphs (Morris et al., 2020; Hu et al., 2020), have recurring structures, but fixed coarsening rules cannot learn from these patterns.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

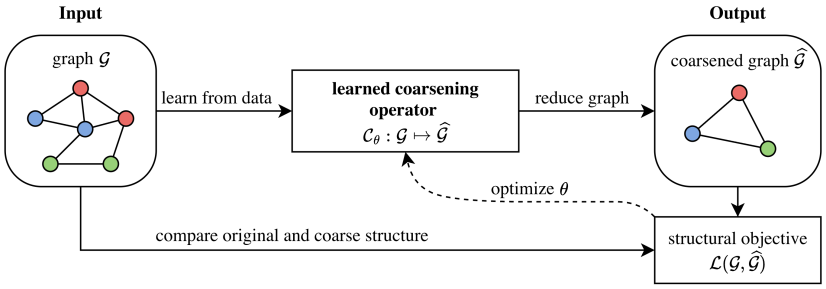


Figure 1: **Learning a coarsening operator.** Given an input graph $\mathcal{G}$, the goal is to learn an operator $\mathcal{C}_{\boldsymbol{\theta}}$ that produces a coarsened graph $\widehat{\mathcal{G}} = \mathcal{C}_{\boldsymbol{\theta}}(\mathcal{G})$. The preservation objective $\mathcal{L}(\mathcal{G}, \widehat{\mathcal{G}})$ compares the original and coarsened graph with respect to the property that should be preserved. Since the loss is computed from $\mathcal{G}$ and $\widehat{\mathcal{G}}$ alone, no reference coarsening or labels are required.

This motivates learning a coarsening operator from a distribution of graphs. GOREN takes a step in this direction by learning coarse edge weights, but leaves the choice of which nodes to merge to a classical algorithm (Cai et al., 2021). DiffPool and OTCoarsening learn the aggregation itself, but allow an input node to contribute to several supernodes through weighted assignments (Ying et al., 2018; Ma & Chen, 2021). These assignments do not enforce a partition into disjoint, connected groups. DiffPool's dense assignments also produce a dense coarse graph. EdgePool uses graph coarsening within a GNN trained for graph classification (Diehl et al., 2019). It trains edge scores through pooled node features, without passing gradients through the discrete coarsening decisions.

We learn the discrete coarsening operator itself, so that the preservation objective guides which nodes to merge and hence the structure of the coarse graph. Each node belongs to exactly one of a prescribed number of connected groups. We train this operator directly from a loss comparing the original graph with the discrete coarse graph it produces. Section A gives a detailed comparison with prior work.

Let $\mathcal{D}$ be a distribution of graphs and let $\mathfrak{C}$ be a class of feasible coarsening operators. Each operator $\mathcal{C} \in \mathfrak{C}$ maps an input graph $\mathcal{G}$ to a coarsened graph $\widehat{\mathcal{G}} = \mathcal{C}(\mathcal{G})$. The loss $\mathcal{L}(\mathcal{G}, \widehat{\mathcal{G}}) \geq 0$ measures how well $\widehat{\mathcal{G}}$ preserves the target property of $\mathcal{G}$. The ideal operator solves

$$\mathcal{C}^\star \in \arg\min_{\mathcal{C} \in \mathfrak{C}} \mathbb{E}_{\mathcal{G} \sim \mathcal{D}} \big[\mathcal{L}(\mathcal{G}, \mathcal{C}(\mathcal{G})) \big]. \tag{1}$$

We parameterize the operator by a graph neural network and learn

$$\boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{G} \sim \mathcal{D}} \big[\mathcal{L}(\mathcal{G}, \mathcal{C}_{\boldsymbol{\theta}}(\mathcal{G})) \big]. \tag{2}$$

Training is unsupervised since $\mathcal{L}$ compares $\mathcal{G}$ with $\widehat{\mathcal{G}}$ directly and no reference coarsening is needed. Changing $\mathcal{L}$ changes which graph property the operator is trained to preserve. Figure 1 illustrates the learning problem.

We require the operator to return a prescribed number of supernodes, each formed from a connected set of original nodes. We construct such an operator using edge contractions. For a graph with n nodes, contracting a forest with $n - \hat{n}$ edges gives exactly $\hat{n}$ supernodes, so we need to learn which edges to contract. We use a graph neural network to score the edges and a greedy algorithm to select a maximum weight forest of the required size. The forest construction enforces connectivity and the target size.

To train the network, we need to pass a gradient through the discrete choice of forest. Small changes to the edge scores usually leave the selected forest and the loss unchanged, so the gradient with respect to the scores is zero almost everywhere. We add noise to the scores to make the loss differentiable in expectation, and use a straight-through backward pass to approximate the gradient from the coarsening matrix to the scores. Every forward pass still produces a discrete coarsening. At inference, we either return the deterministic forest from the unperturbed scores or select the coarsening with the lowest loss from several perturbed candidates. We call these modes *deterministic* and *best-of-K inference*.

We evaluate the method on Laplacian spectral preservation. The low end of the Laplacian spectrum describes connectivity, clusters, and diffusion behavior (von Luxburg, 2007). This objective also allows direct comparison with established spectral coarsening methods (Loukas, 2019; Jin et al., 2020; Chen et al., 2023).

2 Coarsening Operators

We consider undirected, connected, simple graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes, $m = |\mathcal{E}|$ edges, adjacency matrix $\mathbf{A}$, and degree matrix $\mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbf{1})$. The graph Laplacian is $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

Coarsening representation. A coarsening partitions $\mathcal{V}$ into $\hat{n}$ nonempty *contraction sets* $\mathcal{C}_1, \ldots, \mathcal{C}_{\hat{n}}$, each collapsed to a supernode of a coarse graph $\widehat{\mathcal{G}} = (\widehat{\mathcal{V}}, \widehat{\mathcal{E}})$. Its edges record adjacency between contraction sets: for $c \neq d$, the coarse edge $\{c, d\} \in \widehat{\mathcal{E}}$ exists iff there are nodes $i \in \mathcal{C}_c$ and $j \in \mathcal{C}_d$ with $\{i, j\} \in \mathcal{E}$. We represent the partition by a *coarsening matrix* $\mathbf{P} \in \{0, 1\}^{\hat{n} \times n}$ with $\mathbf{P}_{ci} = 1$ iff node $i \in \mathcal{C}_c$. For a requested reduction ratio $r \in (0, 1)$, we set $\hat{n} = \lfloor (1 - r)n \rfloor$ and require $1 \leq \hat{n} < n$. The achieved ratio $1 - \hat{n}/n$ may differ slightly from r because of rounding. We obtain the coarse Laplacian by Galerkin projection,

$$\widehat{\mathbf{L}} \;=\; \mathbf{P}\,\mathbf{L}\,\mathbf{P}^{\top} \;\in\; \mathbb{R}^{\hat{n} \times \hat{n}}, \tag{3}$$

as in spectral graph coarsening (Loukas, 2019). For this binary $\mathbf{P}$, coarse edge weights sum the original edges between contraction sets. We write the loss as $\mathcal{L}(\mathcal{G}, \mathbf{P}) \geq 0$ because $\mathbf{P}$ determines the coarse graph. The loss compares the input and its coarsening, so training needs no reference coarsenings or labels. Section 4.1 specifies the spectral objective used in our experiments.

Coarsening by edge contraction. We require each contraction set to be connected in the original graph and build such partitions by *edge contraction* (Diestel, 2017). Contracting an edge merges its two endpoints, and contracting a set of edges $F \subseteq \mathcal{E}$ merges each connected component of $(\mathcal{V}, F)$ into a supernode.

Each selected edge that joins two previously separate components reduces the number of supernodes by one. Selecting only such edges gives a forest and lets us control the number of supernodes exactly.

Proposition 1 (Number of supernodes after forest contraction)**.** *Let $F \subseteq \mathcal{E}$ and let the contraction sets be the connected components of $(\mathcal{V}, F)$. Contracting F yields exactly $n - |F|$ supernodes, each internally connected, if and only if $(\mathcal{V}, F)$ is acyclic. In particular, the coarsenings to $\hat{n}$ supernodes obtainable by edge contraction are exactly those induced by spanning forests F with $|F| = n - \hat{n}$.*

The proof is given in Section B.

By Proposition 1, every partition into $\hat{n}$ connected contraction sets can be represented by a forest in

$$\mathcal{F}_{\hat{n}} \;\coloneqq\; \big\{\, F \subseteq \mathcal{E} : (\mathcal{V}, F) \text{ acyclic},\ |F| = n - \hat{n} \,\big\}, \qquad F \mapsto \mathbf{P}_F, \tag{4}$$

where $\mathbf{P}_F$ assigns each node to its connected component in $(\mathcal{V}, F)$. Different forests may induce the same partition. Learning the operator amounts to learning which forest to select from $\mathcal{F}_{\hat{n}}$.

3 Deep Graph Coarsening

We represent the operator $\mathcal{C}_{\boldsymbol{\theta}}$ from Section 1 by its coarsening matrix $f_{\boldsymbol{\theta}}(\mathcal{G}) = \mathbf{P}$. We use a graph neural network to score the edges and a greedy algorithm to select a maximum weight forest of the required size (Sections 3.1 and 3.2). We train the scores using a loss on the resulting coarse graph. Score perturbations and a straight-through backward pass provide a training signal through the discrete forest selection and cluster assignments (Section 3.3). Figure 4 summarizes the procedure.

3.1 Scoring network

We compute the edge scores with a standard message passing network $\psi_{\boldsymbol{\theta}}$. Because we do not use node attributes, we construct structural input features for each node. We concatenate its local degree profile (Cai & Wang, 2018) with the first k_{PE} nontrivial Laplacian eigenvectors (Dwivedi

& Bresson, 2020) and map the result through an MLP to obtain the initial node embedding $\mathbf{h}_v^{(0)}$. These features can be precomputed once for the entire dataset. The positional encodings improve spectral preservation in our experiments, but are not required by the method. We then apply L graph attention layers (Veličković et al., 2018) to obtain node embeddings $\mathbf{h}_u$. For each edge $e = \{u, v\}$, the readout is symmetric in its two endpoints,

$$\begin{aligned} \tilde{s}_e &= \mathrm{MLP}\big([\,\mathbf{h}_u + \mathbf{h}_v \,\|\, |\mathbf{h}_u - \mathbf{h}_v|\,]\big), \\ s_e &= \tilde{s}_e - \frac{1}{m}\sum_{e' \in \mathcal{E}} \tilde{s}_{e'}. \end{aligned} \tag{5}$$

We center the edge scores per graph to remove an arbitrary offset without changing the selected forest. Section D gives the architecture details.

3.2 Coarsening as a maximum weight forest

The scoring network assigns $\mathbf{s} = \psi_{\boldsymbol{\theta}}(\mathcal{G}) \in \mathbb{R}^m$, where a larger s_e favors contracting edge e. For a requested reduction ratio r, we aim to retain $\hat{n}(r) = \lfloor (1-r)n \rfloor$ supernodes. By Proposition 1 and Equation (4), every connected coarsening with $\hat{n}(r)$ supernodes is represented by a forest in $\mathcal{F}_{\hat{n}(r)}$. We select a forest with maximum total score,

$$F^\star(\mathcal{G}, \mathbf{s}, r) \;=\; \operatorname*{arg\,max}_{F \in \mathcal{F}_{\hat{n}(r)}} \sum_{e \in F} s_e \;=\; \operatorname*{arg\,max}_{F \in \mathcal{F}_{\hat{n}(r)}} \langle \mathbf{s}, \boldsymbol{\chi}_F \rangle\,, \tag{6}$$

where $\boldsymbol{\chi}_F \in \{0,1\}^m$ is the indicator of F. The connected components of a forest F define its contraction sets. Under any fixed ordering of these components, row c of $\mathbf{P}_F$ indicates which nodes belong to the cth component. The coarsening operator returns this matrix for the selected forest,

$$\Pi(\mathcal{G}, \mathbf{s}, r) \;=\; \mathbf{P}_{F^\star(\mathcal{G}, \mathbf{s}, r)} \;\in\; \{0,1\}^{\hat{n}(r) \times n}. \tag{7}$$

The reduction ratio fixes the number of supernodes, and the scores determine which nodes are merged. Algorithm 1 solves the forest selection problem exactly.

Proposition 2 (Greedy optimality)**.** *For any scores $\mathbf{s} \in \mathbb{R}^m$ and integer $1 \le \hat{n} < n$, the forest selected by Algorithm 1 lies in $\mathcal{F}_{\hat{n}}$ and maximizes $\langle \mathbf{s}, \boldsymbol{\chi}_F \rangle$ over $\mathcal{F}_{\hat{n}}$.*

The proof and complete pseudocode are given in Section B.

After sorting edges by score, Algorithm 1 uses union–find to select the forest and constructs its coarsening matrix. The learned operator is $f_{\boldsymbol{\theta}}(\mathcal{G}) = \Pi(\mathcal{G}, \psi_{\boldsymbol{\theta}}(\mathcal{G}), r)$.

Training requires a gradient from the loss on $\mathbf{P}$ back to the edge scores. Two discrete operations block this path: selecting the forest indicator $\boldsymbol{\chi}$ and converting its connected components into $\mathbf{P}$. The resulting map from scores to $\mathbf{P}$ is piecewise constant, so the loss gradient with respect to the scores is zero almost everywhere. We perturb the scores to obtain a smooth expected loss and use a straight-through estimator to pass a training signal through both discrete operations.

3.3 Training with perturbed scores

We draw noise from a distribution μ on $\mathbb{R}^m$ and perturb the scores before applying the same discrete operator,

$$\Pi(\mathcal{G}, \mathbf{s} + \varepsilon\mathbf{Z}, r) = \mathbf{P}_{\arg\max_{F \in \mathcal{F}_{\hat{n}(r)}} \langle \mathbf{s} + \varepsilon\mathbf{Z}, \boldsymbol{\chi}_F \rangle}, \qquad \mathbf{Z} \sim \mu. \tag{8}$$

Every noise sample still produces a valid coarsening through Algorithm 1. Averaging the resulting loss over perturbations and graphs from the empirical training distribution $\mathcal{D}$ gives the objective

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{G} \sim \mathcal{D}} \mathbb{E}_{\mathbf{Z} \sim \mu} \left[\mathcal{L}\left(\mathcal{G}, \Pi(\mathcal{G}, \psi_{\boldsymbol{\theta}}(\mathcal{G}) + \varepsilon\mathbf{Z}, r)\right) \right]. \tag{9}$$

Setting $\varepsilon = 0$ recovers the deterministic operator. For $\varepsilon > 0$, the following result shows that the expected loss is differentiable with respect to the scores.

Proposition 3 (Smoothing)**.** *Fix $\mathcal{G}$ and assume that $\mathcal{L}$ is finite on every admissible coarsening. Let μ be a distribution on $\mathbb{R}^m$ with positive C^1 density p_μ and $\nabla p_\mu \in L^1(\mathbb{R}^m)$, let $\mathbf{Z} \sim \mu$, and let $\varepsilon > 0$. Then*

$$J_\varepsilon(\mathbf{s}) = \mathbb{E}_{\mathbf{Z} \sim \mu} \left[\mathcal{L}\big(\mathcal{G}, \Pi(\mathcal{G}, \mathbf{s} + \varepsilon\mathbf{Z}, r)\big) \right]$$

is differentiable in $\mathbf{s}$*.*

Proof. See Section C.1. □

For training, we use a straight-through estimator (Bengio et al., 2013). We differentiate the loss with respect to the sampled coarsening matrix $\mathbf{P}$ (Sections 4.1 and E) and pass this signal back to the edge scores using approximate backward steps through the two discrete operations. The expected straight-through update need not equal the gradient of the smoothed objective. Section C.1 derives the exact gradient and explains this distinction.

In each training forward pass, we sample $\mathbf{Z} \sim \mathrm{Gumbel}(0,1)^m$, construct

$$\mathbf{P} = \Pi(\mathcal{G}, \mathbf{s} + \varepsilon\mathbf{Z}, r),$$

and evaluate $\mathcal{L}(\mathcal{G}, \mathbf{P})$.

Straight-through estimator. The discrete operator consists of two maps. The selection map $\mathbf{s} \mapsto \boldsymbol{\chi}$ chooses a forest indicator through Equation (6), while the grouping map $\boldsymbol{\chi} \mapsto \mathbf{P}$ converts the connected components of that forest into cluster assignments. The backward pass traverses these maps in reverse and replaces their missing derivatives separately.

First, the projection surrogate approximates the backward pass through the grouping map. Let

$$\mathbf{U} = \frac{\partial \mathcal{L}}{\partial \mathbf{P}}, \qquad \rho_i = \|\mathbf{U}_{:,i}\|_2, \qquad q_{\{i,j\}} = \rho_i + \rho_j.$$

The column $\mathbf{U}_{:,i}$ records the sensitivity of the loss to the cluster assignment of node i. Its norm gives a scalar signal that is independent of the arbitrary ordering of the supernodes. Since contracting an edge $\{i,j\}$ directly affects the assignments of both endpoints, $q_{\{i,j\}}$ provides a local surrogate for the sensitivity of selecting that edge. We normalize this signal by its graphwise mean,

$$\tilde{\mathbf{q}} = \frac{\mathbf{q}}{\frac{1}{m}\sum_{e' \in \mathcal{E}} q_{e'} + \delta} \approx \frac{\partial \mathcal{L}}{\partial \boldsymbol{\chi}}, \tag{10}$$

where $\delta > 0$ is a small numerical constant. Taking norms discards gradient direction, so this is a heuristic signal rather than the derivative of component assignment. Both the base estimator and the optional forest mask use it.

Second, the selection surrogate approximates the missing derivative $\partial\boldsymbol{\chi}/\partial\mathbf{s}$ of the forest selection in Equation (6) through a differentiable top-k relaxation for sampling without replacement (Plötz & Roth, 2018; Xie & Ermon, 2019). Let $k = n - \hat{n}(r)$ and initialize $\boldsymbol{\alpha}^{(1)} = \mathbf{s} + \varepsilon\mathbf{Z}$. With Gumbel perturbations, this yields a Gumbel–Softmax relaxation (Jang et al., 2017) with k differentiable selections,

$$\mathbf{p}^{(t)} = \mathrm{softmax}(\boldsymbol{\alpha}^{(t)}/\tau), \qquad \boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + \log(1 - \mathbf{p}^{(t)}), \qquad \tilde{\boldsymbol{\chi}} = \sum_{t=1}^{k} \mathbf{p}^{(t)}. \tag{11}$$

The relaxed vector $\tilde{\boldsymbol{\chi}}$ does not enforce acyclicity. We use it only in the backward pass, while the forward pass still selects a forest. Combining the grouping and selection surrogates gives

$$\frac{\partial \mathcal{L}}{\partial \mathbf{s}} \approx \left(\frac{\partial \tilde{\boldsymbol{\chi}}}{\partial \mathbf{s}}\right)^{\top} \tilde{\mathbf{q}}. \tag{12}$$

The temperature τ controls the sharpness of the relaxation. Setting $\varepsilon = 0$ removes the perturbation and gives the deterministic softmax relaxation used in the corresponding ablation. Positive ε gives the Gumbel–Softmax top-k relaxation used by our base estimator (Section F).

Forest mask. At high reduction ratios, the dense top-k relaxation can assign gradient to edges that the greedy solver rejects because they would create cycles. We therefore optionally restrict the backward signal to the selected forest:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{s}} \approx \boldsymbol{\chi} \odot \tilde{\mathbf{q}}. \tag{13}$$

This replaces the relaxed selection Jacobian by $\mathrm{diag}(\boldsymbol{\chi})$. The forward pass and grouping surrogate are unchanged, while the backward signal is restricted to selected edges. We evaluate this heuristic at high reduction ratios in Section 4.3.

3.4 INFERENCE

At inference, setting $\varepsilon = 0$ yields the maximum weight forest from Equation (6), hence one coarsening per graph. We call this *deterministic inference*. With $\varepsilon > 0$, we instead generate K perturbed candidates using Equation (8), include the deterministic forest, and select the candidate with the smallest selection loss. We distinguish *best-of-K inference (training loss)*, which selects by the loss used for training, from *best-of-K inference (evaluation metric)*, which selects by the metric used for evaluation. In our experiments, these are the quadratic loss and eigenerror, respectively (Section 4.1). In both modes, K counts the perturbed candidates, and the deterministic candidate is included in addition.

Including the deterministic candidate guarantees that selection is no worse under the selection loss than deterministic inference. Adding candidates to an existing set can only decrease or preserve its minimum loss. These guarantees need not hold for a different evaluation metric.

Computational complexity. With node features available, a fixed network architecture, and a sparse coarsening matrix, edge scoring costs $O(n + m)$. Constructing the maximum weight forest requires $O(m \log m)$ for sorting, followed by $O(m\alpha(n) + n)$ for union–find and cluster construction. A deterministic coarsening therefore costs $O(n + m \log m)$, independently of the objective $\mathcal{L}$. Training, sampled inference, and feature preprocessing are analyzed in Section G.

Learned loss model. One alternative is to predict the true coarsening loss from the edge scores with a differentiable network and train the scorer through this surrogate. In our experiments, its gradients do not reliably improve the scorer. The model, alternating training scheme, and results are given in Sections I and J.

4 EXPERIMENTS

We compare the learned operator with classical coarsening methods and GOREN on Laplacian spectral preservation at the same requested reduction ratios. We also compare deterministic and best-of-K inference and examine the effect of the forest mask.

4.1 EXPERIMENTAL SETUP

The smallest Laplacian eigenvalues reflect graph connectivity, and their eigenvectors are used in spectral clustering (von Luxburg, 2007). We therefore evaluate preservation of the unnormalized Laplacian spectrum using the relative *eigenerror*

$$\mathcal{L}_{\text{eig}}(\mathcal{G}, \mathbf{P}) = \frac{1}{k} \sum_{i=1}^{k} \frac{\left| \hat{\lambda}_i - \lambda_i \right|}{\lambda_i}, \tag{14}$$

where λ_i and $\hat{\lambda}_i$ are the ordered nonzero eigenvalues of the original Laplacian $\mathbf{L}$ and the Galerkin coarse Laplacian $\widehat{\mathbf{L}} = \mathbf{P}\mathbf{L}\mathbf{P}^\top$ from Equation (3), respectively. We use $k = 10$ unless stated otherwise, which requires at least eleven coarse nodes.

Because eigenerror is not generally differentiable with respect to the coarsening matrix, we train with the quadratic form loss of Cai et al. (2021). Let $\mathbf{f}_i$ be a unit eigenvector associated with λ_i. The map $\mathbf{P}^{\mp} = (\mathbf{P}^{+})^\top = \Gamma^{-1}\mathbf{P}$ averages signals on the original graph within each contraction set. The loss is

$$\mathcal{L}_{\text{quad}}(\mathcal{G}, \mathbf{P}) = \frac{1}{k} \sum_{i=1}^{k} \left| \mathbf{f}_i^\top \mathbf{L} \mathbf{f}_i - (\mathbf{P}^{\mp}\mathbf{f}_i)^\top \widehat{\mathbf{L}} (\mathbf{P}^{\mp}\mathbf{f}_i) \right|. \tag{15}$$

The loss compares the energy of each eigenvector on the original graph with that of its average on the coarse graph. By the Galerkin identity, the coarse term also equals the energy on the original graph of $\mathbf{P}^\top\mathbf{P}^{\mp}\mathbf{f}_i$, the reconstruction that is constant within each contraction set. The loss provides matrix gradients without differentiating coarse eigenvalues. We train all final models of our operator with this quadratic loss and evaluate them by eigenerror. Sections E and P describe the Rayleigh variant and the validation comparison between the two losses.

The final benchmark uses nine graph families: PROTEINS (Morris et al., 2020), ENZYMES (Morris et al., 2020), COLLAB (Morris et al., 2020), PUBMED (Namata et al., 2012; Yang et al., 2016), AMAZON PHOTO (Shchur et al., 2019), AMAZON COMPUTERS (Shchur et al., 2019), SBM (Holland et al., 1983), Erdős–Rényi (ER) graphs (Erdős & Rényi, 1959), and BA (Barabási & Albert, 1999). Each method is evaluated at $r \in \{0.25, 0.50, 0.75\}$. For each dataset and reduction ratio, we select the classical method with the lowest validation eigenerror and report the test result of that fixed method as *best classical*. The candidates are Heavy Edge (Karypis & Kumar, 1998), Algebraic JC (Ron et al., 2011), Affinity GS (Livne & Brandt, 2012), Kron reduction (Dörfler & Bullo, 2013), Variation Edges (Loukas, 2019), and Variation Neighborhoods (Loukas, 2019). We also evaluate GOREN (Cai et al., 2021), which learns coarse edge weights after a classical algorithm fixes the contraction sets. We select its classical coarsening method on validation data and train its edge-weight model with the same quadratic loss as our operator. Section M also reports the comparison with GOREN trained using its original Rayleigh objective. Section A discusses GOREN and other learned reduction methods. We evaluate three inference modes using the same trained scorer. *Deterministic inference* returns the maximum weight forest without score perturbations. *Best-of-K inference (training loss)* chooses among perturbed candidates by the quadratic loss, while *best-of-K inference (evaluation metric)* chooses by eigenerror. Both sampled modes include the deterministic output. We select the checkpoint, perturbation scale, and candidate budget on validation data before test evaluation. For best-of-K inference (evaluation metric), all final base-STE configurations use $K = 128$ perturbed candidates. Neither selection rule requires labels, and best-of-K inference (training loss) avoids computing the coarse eigenvalues for each candidate.

We also compare against random contraction, both averaged over draws and with selection of the lowest-eigenerror forest among 128 draws. This uses the same number of sampled forests as best-of-K inference (evaluation metric), which also includes the learned deterministic output. It tests whether learned scores improve on random candidate search. Methods receive the same requested reduction ratio. Our forest construction reaches the prescribed size exactly, whereas some classical methods retain more nodes than requested. Since computing $k = 10$ nonzero eigenvalues requires $\hat{n} > k$, we compare all methods on the common subset of graphs whose prescribed target size satisfies this condition. Sections K and L give the dataset statistics, the eligibility rule, and the full results.

4.2 SPECTRAL PRESERVATION

With best-of-K inference (evaluation metric), the learned operator has lower mean test eigenerror than the best classical baseline in 24 of 27 dataset–ratio settings. Figure 2 separates this result from deterministic inference. The fraction of datasets on which the deterministic output has lower error is 5/9, 6/9, and 9/9 at $r = 0.25, 0.50, 0.75$, respectively. Best-of-K inference (evaluation metric) increases these fractions to 7/9, 8/9, and 9/9.

For best-of-K inference (evaluation metric) at $r = 0.25$, the mean relative eigenerror reduction across datasets is 12.8% (Table 2). The best classical baseline still performs better on SBM and BA. At $r = 0.50$, the mean reduction is 37.0%, and at $r = 0.75$ the learned operator has lower error on all nine datasets. These comparisons concern mean errors and do not establish statistical significance.

Comparison with GOREN. With both models trained using the quadratic loss, our operator under deterministic inference has lower test eigenerror than GOREN in 16 of 27 settings, including six of nine datasets at both $r = 0.50$ and 0.75. The comparison separates two components of graph coarsening. GOREN optimizes coarse edge weights after a classical method has fixed the contraction sets, whereas our operator learns the contractions themselves. The majority advantage shows that weight optimization alone does not consistently compensate for a fixed coarsening structure, and that learning the contraction decisions is important at moderate and high reduction. However, the two approaches are complementary. GOREN's weight optimization could be applied to the contraction sets produced by our operator and may provide further improvements. The complete results and the comparison with GOREN's original Rayleigh objective are reported in Section M.

Figure 3 compares the three inference modes with random candidate search, and measures the improvement from sampling after the scorer has been trained. Best-of-K inference (evaluation metric)

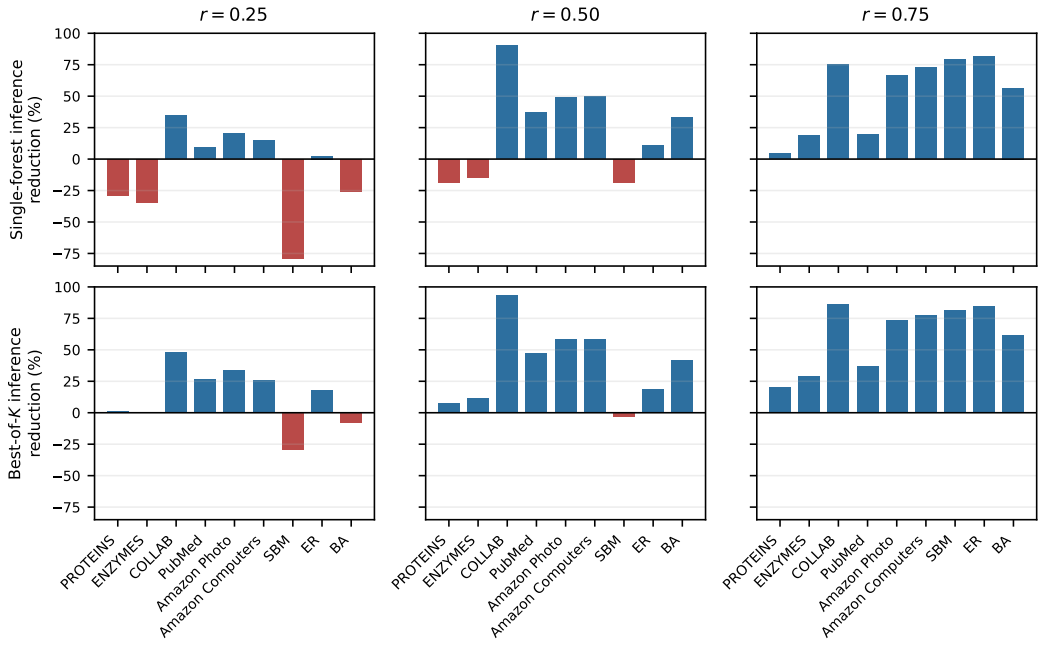


Figure 2: **Relative spectral improvement over best classical coarsening.** Bars show percentage eigenerror reduction relative to the best classical method selected on validation. Positive values are better. Top: deterministic inference. Bottom: best-of-K inference (evaluation metric) with $K = 128$ perturbed forests plus the deterministic forest.

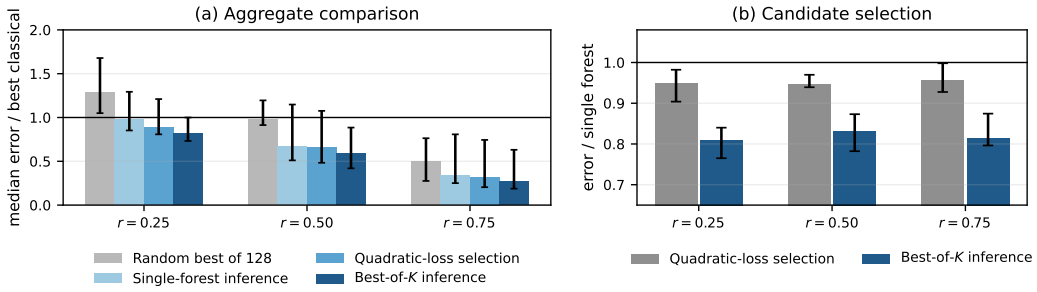


Figure 3: **Learning and candidate selection.** Left: median error relative to the classical method selected on validation, with interquartile ranges across datasets. Random best of 128 also samples 128 forests, but generates them without a learned scorer. Right: best-of-K inference (training loss) with K selected on validation and best-of-K inference (evaluation metric) with $K = 128$ perturbed forests plus the deterministic forest, both relative to deterministic inference.

cannot increase eigenerror relative to the included deterministic candidate. Best-of-K inference (training loss) has no corresponding guarantee for eigenerror, since it selects by a different loss.

Best-of-K inference (evaluation metric) has lower eigenerror than Random best of 128 in 26/27 settings. The learned scores therefore improve on random search at the same sampled-candidate budget. Sampling also improves on the scorer's deterministic output. Across datasets, the median relative reductions in eigenerror from best-of-K inference (evaluation metric) are 19.1%, 16.9%, and 18.7% at the three reduction ratios. Best-of-K inference (training loss) gives smaller median relative reductions of 5.2%, 5.4%, and 4.5%, respectively. Section N gives the full comparison.

4.3 Effect of the forest mask

The forest mask restricts the backward signal to edges in the selected forest. At $r = 0.75$, we compare it with the base estimator, using best-of-K inference (evaluation metric) for both. The mask lowers mean eigenerror on PROTEINS, ENZYMES, and AMAZON PHOTO. The base estimator performs at least as well on the other six datasets, so we retain it as the default. Section O gives the complete comparison.

4.4 Additional analyses

The training ablation in Section H compares updates with and without score perturbation. Perturbation reduces instability in the plotted runs, and the forest mask gives a steadier curve at high reduction. The learned surrogate loss in Section J does not reliably improve the scorer under joint training.

Further experiments examine transfer across reduction ratios, spectral cutoffs, graph sizes, and datasets (Section R), and report eigenerror over a finer grid of reduction ratios (Section T). The learned advantage is less consistent for normalized spectra and downstream clustering (Sections U and V). On larger DD and REDDIT-BINARY graphs, best-of-K inference (evaluation metric) substantially lowers the error. It reverses the deterministic gap on DD, whereas the best classical baseline remains stronger on REDDIT-BINARY (Section S).

5 Conclusion

We learn a discrete graph coarsening operator directly from a preservation objective. A graph neural network scores edges for contraction, and a maximum weight forest gives connected contraction sets with exact control over the number of supernodes. We train with perturbed scores and a straight-through estimator, while every forward pass retains the discrete forest construction.

For Laplacian spectral preservation, the learned operator under deterministic inference has a lower mean eigenerror than the classical baseline selected on validation in most benchmark settings. Best-of-K inference (evaluation metric) further reduces eigenerror, at the cost of evaluating several coarsenings.

The gains do not extend uniformly to the normalized spectrum or downstream clustering. In particular, preserving eigenvalues does not ensure that the eigenvectors used for clustering are preserved. An open question for future work is how to train the operator to preserve the relevant eigenspaces and whether doing so improves downstream clustering.

AI use statement

We used generative AI tools to support literature retrieval and discovery, research ideation, the implementation and analysis of experiments, the development and verification of mathematical proofs, and the drafting and linguistic revision of parts of the manuscript. The authors reviewed and revised all AI-assisted text and derivations, tested AI-assisted code, verified the experimental results, and checked identified references against the original sources. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

Reproducibility statement

To support reproducibility, the main paper specifies the learned coarsening operator, its training procedure, and the objectives used for training and evaluation. The appendix provides complete proofs (Sections B and C), implementation details (Sections D to G), and the full experimental protocol (Section K). This includes dataset construction, fixed train/validation/test splits, baseline implementations, hyperparameters, model selection, and inference settings. Complete numerical results and additional evaluations are reported in Section L.

REFERENCES

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286 (5439):509–512, 1999. doi: 10.1126/science.286.5439.509.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.

Andreas Bergmeister, Karolis Martinkus, Nathanaël Perraudin, and Roger Wattenhofer. Efficient and scalable graph generation through iterative local expansion. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=2XkTz7gdpc.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable perturbed optimizers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification. *ArXiv*, abs/1811.03508, 2018. URL https://api.semanticscholar.org/CorpusID:53214254.

Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uxpzitPEooJ.

Yifan Chen, Rentian Yao, Yun Yang, and Jie Chen. A Gromov-Wasserstein geometric view of spectrum-preserving graph coarsening. In *International Conference on Machine Learning (ICML)*, 2023.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 13260–13271. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/99cad265a1768cc2dd013f0e740300ae-Paper.pdf.

Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11): 1944–1957, 2007. doi: 10.1109/TPAMI.2007.1115.

Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. Towards graph pooling by edge contraction. In *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Berlin, Heidelberg, 5th edition, 2017. ISBN 978-3-662-53622-3. doi: 10.1007/978-3-662-53622-3.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *ArXiv*, abs/2012.09699, 2020.

Florian Dörfler and Francesco Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):150–163, 2013. doi: 10.1109/TCSI.2012.2215780.

Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

Paul Erdős and Alfréd Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6: 290–297, 1959. doi: 10.5486/PMD.1959.6.3-4.12.

Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983. doi: 10.1016/0378-8733(83)90021-7.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for machine learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/fb60d411a5c5b72b2e7d3527cfc84fd0-Paper.pdf.

Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964. doi: 10.1214/aoms/1177703732. URL https://doi.org/10.1214/aoms/1177703732.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Yu Jin, Andreas Loukas, and Joseph JaJa. Graph coarsening with preserved spectral properties. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1998. doi: 10.1137/S1064827595287997.

Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

Oren Livne and Achi Brandt. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Journal on Scientific Computing*, 34:B499–B522, 08 2012. doi: 10.1137/110843563.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.*, 20(116): 1–42, 2019.

Tengfei Ma and Jie Chen. Unsupervised learning of graph hierarchical abstractions with differentiable coarsening and optimal transport. In *AAAI Conference on Artificial Intelligence*, 2021.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *J. Artif. Int. Res.*, 80, September 2024. ISSN 1076-9757. doi: 10.1613/jair.1.15320.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020. URL https://api.semanticscholar.org/CorpusID:220633407.

Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven active surveying for collective classification. In *Proceedings of the Workshop on Mining and Learning with Graphs*, Edinburgh, Scotland, UK, 2012. URL https://people.cs.vt.edu/~bhuang/papers/namata-mlg12.pdf.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. 32, 2018. doi: 10.1609/aaai.v32i1.11671.

Tobias Plötz and Stefan Roth. Neural nearest neighbors networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/f0e52b27a7a5d6a1a87373dffa53dbe5-Paper.pdf.

Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011. doi: 10.1137/100791142.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of Graph Neural Network evaluation, 2019. URL https://arxiv.org/abs/1811.05868.

David I. Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Trans. Signal Process.*, 64(8):2119–2134, 2016. doi: 10.1109/TSP.2015.2512529.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

Yan Wang, Wenju Hou, Nan Sheng, Ziqi Zhao, Jialin Liu, Lan Huang, and Juexin Wang. Graph pooling in graph neural networks: methods and their applications in omics studies. *Artificial Intelligence Review*, 57, 2024. URL https://api.semanticscholar.org/CorpusID:273190347.

Sang Michael Xie and Stefano Ermon. Reparameterizable subset sampling via continuous relaxations. In *International Joint Conference on Artificial Intelligence*, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5453–5462. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/xu18c.html.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*, pp. 40–48, 2016.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley J. Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Skh4jRcKQ.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Neural Information Processing Systems*, 2018. URL https://api.semanticscholar.org/CorpusID:49420315.

APPENDIX

The appendix provides related work, proofs, implementation details, and the full experimental results. Figure 4 summarizes the method.

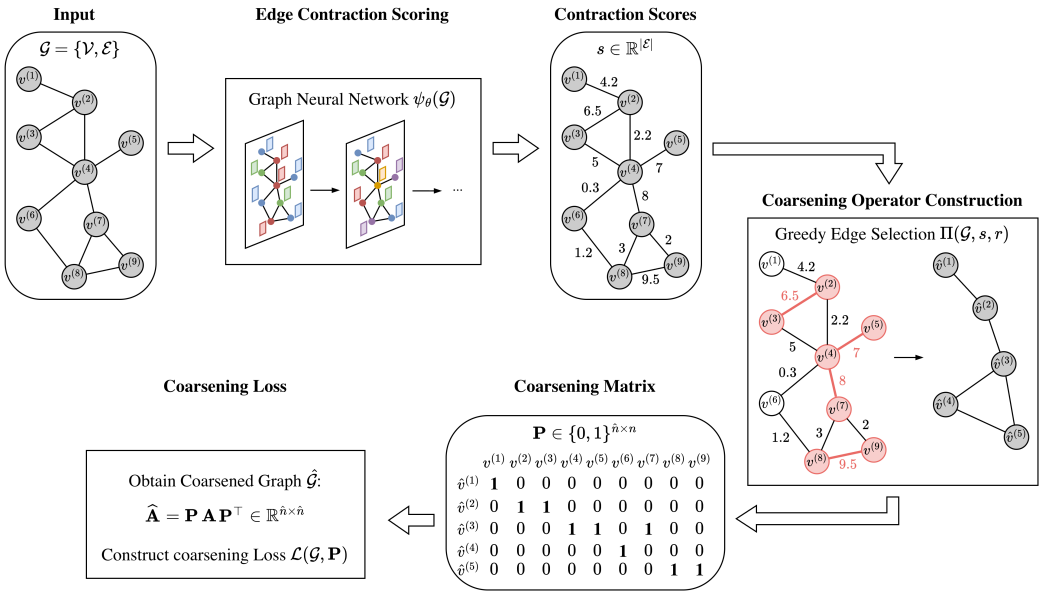


Figure 4: Overview of the deep graph coarsening pipeline. The scoring network $\psi_{\boldsymbol{\theta}}$ assigns contraction scores to the edges, the greedy operator Π constructs the coarsening matrix from the selected contraction forest, and this matrix defines the coarse graph used by the loss.

A RELATED WORK

Classical graph coarsening. Heavy edge matching (Karypis & Kumar, 1998), algebraic distance (Ron et al., 2011), and affinity (Livne & Brandt, 2012) choose contractions using fixed measures of edge strength or similarity. Kron reduction eliminates vertices through a Schur complement of the Laplacian (Dörfler & Bullo, 2013; Shuman et al., 2016). Local variation chooses contraction sets by minimizing a restricted spectral variation cost (Loukas, 2019). These methods use graph structure to make coarsening decisions, but their rules are not learned from a graph distribution.

Learning graph coarsening. GOREN (Cai et al., 2021) learns coarse edge weights after a classical algorithm has fixed the aggregation. We instead learn edge scores that determine the contraction sets themselves. OTCoarsening (Ma & Chen, 2021) also learns coarsening without labels. It selects coarse nodes and learns weighted aggregation using an optimal transport objective. Our operator partitions the original nodes through edge contractions, so each node belongs to exactly one connected contraction set.

Graph pooling and edge contraction. Graph pooling often learns graph reduction as part of a supervised GNN (Wang et al., 2024). DiffPool learns dense soft assignments (Ying et al., 2018), which do not enforce connected contraction sets. EdgePool learns edge scores and greedily contracts a matching (Diehl et al., 2019). It trains the scores by multiplying them into the pooled node features. We evaluate a preservation loss on the resulting coarse graph and use a surrogate backward pass through the discrete partition. Selecting a forest also lets us prescribe the number of supernodes directly, rather than letting the size of a matching determine the reduction at each layer.

Learning through discrete decisions. Straight-through estimators use a hard forward computation and substitute a surrogate derivative in the backward pass (Bengio et al., 2013; Yin et al., 2019).

Our training rule combines this approach with perturbed edge scores. Perturbation makes the expected loss differentiable (Berthet et al., 2020), while soft top-k selection (Plötz & Roth, 2018; Xie & Ermon, 2019) with Gumbel noise (Jang et al., 2017) supplies one part of our backward surrogate. We also evaluate an alternative that learns a differentiable model of the coarsening loss and uses its gradients to train the scorer (Mandi et al., 2024).

B Forest construction

B.1 Proof of Proposition 1

Proof. A graph on n nodes with $|F|$ edges has $n - |F| + c$ connected components, where c is its number of independent cycles; this equals $n - |F|$ iff $c = 0$, i.e. iff $(\mathcal{V}, F)$ is a forest. Each component, being connected, is an internally connected contraction set. Choosing $|F| = n - \hat{n}$ gives $\hat{n}$ components. □

B.2 Greedy construction and proof of Proposition 2

Algorithm 1 gives the complete greedy construction used by the forward operator.

Algorithm 1: Constructing the coarsening matrix

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, edge scores $\mathbf{s} \in \mathbb{R}^m$, target size $\hat{n}$
Output: coarsening matrix $\mathbf{P}$

$F \leftarrow \emptyset$; initialize the partition $\mathcal{U} \leftarrow \{\{v\} : v \in \mathcal{V}\}$
foreach *edge* $e = \{u, v\}$ *in decreasing order of* s_e **do**
  if $|F| = n - \hat{n}$ **then break**
  if $\mathrm{Find}_{\mathcal{U}}(u) \neq \mathrm{Find}_{\mathcal{U}}(v)$ **then**
    $F \leftarrow F \cup \{e\}$; $\mathrm{Union}_{\mathcal{U}}(u, v)$
Write the final partition as $\mathcal{U} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{\hat{n}}\}$
$\mathbf{P} \leftarrow \mathbf{0} \in \{0, 1\}^{\hat{n} \times n}$
for $c = 1, \ldots, \hat{n}$ **do**
  foreach *node* $i \in \mathcal{C}_c$ **do**
    $\mathbf{P}_{ci} \leftarrow 1$
return $\mathbf{P}$

Proof of Proposition 2. Process the edges in decreasing score and keep each one that joins two different components, stopping once $n - \hat{n}$ edges are kept (Algorithm 1). This is Kruskal's algorithm (Kruskal, 1956) stopped early, and it returns a maximum weight forest of the target size: at every step the kept set is the heaviest acyclic set of its size, a standard exchange argument (Edmonds, 1971). Connectivity of $\mathcal{G}$ guarantees a forest of size $n - \hat{n}$ exists. □

C Differentiability of the expected loss

C.1 Proof of Proposition 3

Fix $\mathcal{G}$ and r. Write the loss as a function of the perturbed score vector,

$$\ell(\mathbf{y}) = \mathcal{L}(\mathcal{G}, \Pi(\mathcal{G}, \mathbf{y}, r)).$$

For fixed $\mathcal{G}$ and r, the admissible forest family $\mathcal{F}_{\hat{n}(r)}$ is finite. Hence Π can return only finitely many coarsening matrices. By the finite-loss assumption in Proposition 3, ℓ is bounded; a fixed tie-breaking rule makes it measurable. Let $|\ell(\mathbf{y})| \leq M$ for all $\mathbf{y}$. Ties in the maximizer may be resolved by any fixed rule; they occur only on score hyperplanes and therefore have probability zero under the continuous perturbation density.

With this notation,

$$J_\varepsilon(\mathbf{s}) = \mathbb{E}_{\mathbf{Z} \sim \mu}\big[\ell(\mathbf{s} + \varepsilon \mathbf{Z})\big] = \int_{\mathbb{R}^m} \ell(\mathbf{s} + \varepsilon \mathbf{z})\, p_\mu(\mathbf{z})\, d\mathbf{z}.$$

Substitute $\mathbf{y} = \mathbf{s} + \varepsilon\mathbf{z}$, so $\mathbf{z} = (\mathbf{y} - \mathbf{s})/\varepsilon$ and $d\mathbf{z} = \varepsilon^{-m} d\mathbf{y}$. Then

$$J_\varepsilon(\mathbf{s}) = \int_{\mathbb{R}^m} \ell(\mathbf{y})\, \varepsilon^{-m} p_\mu\left(\frac{\mathbf{y} - \mathbf{s}}{\varepsilon}\right) d\mathbf{y}. \tag{16}$$

The dependence on $\mathbf{s}$ is now only in the shifted density. Since ℓ is bounded and $\nabla p_\mu \in L^1(\mathbb{R}^m)$, the translated density is differentiable in L^1, which justifies differentiating under the integral. For coordinate j,

$$\begin{aligned}
\frac{\partial J_\varepsilon(\mathbf{s})}{\partial s_j} &= \int_{\mathbb{R}^m} \ell(\mathbf{y})\, \frac{\partial}{\partial s_j}\left[\varepsilon^{-m} p_\mu\left(\frac{\mathbf{y} - \mathbf{s}}{\varepsilon}\right)\right] d\mathbf{y} \\
&= \int_{\mathbb{R}^m} \ell(\mathbf{y})\, \varepsilon^{-m} \sum_{k=1}^{m} \frac{\partial p_\mu}{\partial z_k}\left(\frac{\mathbf{y} - \mathbf{s}}{\varepsilon}\right) \frac{\partial}{\partial s_j}\left(\frac{y_k - s_k}{\varepsilon}\right) d\mathbf{y} \\
&= -\frac{1}{\varepsilon} \int_{\mathbb{R}^m} \ell(\mathbf{y})\, \varepsilon^{-m} \frac{\partial p_\mu}{\partial z_j}\left(\frac{\mathbf{y} - \mathbf{s}}{\varepsilon}\right) d\mathbf{y}.
\end{aligned}$$

Because p_μ is positive, we may write $\partial p_\mu / \partial z_j = p_\mu\, \partial \log p_\mu / \partial z_j$. Thus

$$\frac{\partial J_\varepsilon(\mathbf{s})}{\partial s_j} = \frac{1}{\varepsilon} \int_{\mathbb{R}^m} \ell(\mathbf{y})\, \varepsilon^{-m} p_\mu\left(\frac{\mathbf{y} - \mathbf{s}}{\varepsilon}\right) \left[-\frac{\partial}{\partial z_j} \log p_\mu\left(\frac{\mathbf{y} - \mathbf{s}}{\varepsilon}\right)\right] d\mathbf{y}.$$

Substituting back $\mathbf{y} = \mathbf{s} + \varepsilon\mathbf{z}$ gives

$$\begin{aligned}
\frac{\partial J_\varepsilon(\mathbf{s})}{\partial s_j} &= \frac{1}{\varepsilon} \int_{\mathbb{R}^m} \ell(\mathbf{s} + \varepsilon\mathbf{z}) \left[-\frac{\partial}{\partial z_j} \log p_\mu(\mathbf{z})\right] p_\mu(\mathbf{z})\, d\mathbf{z} \\
&= \frac{1}{\varepsilon} \mathbb{E}_{\mathbf{Z} \sim \mu}\left[\ell(\mathbf{s} + \varepsilon\mathbf{Z}) \left(-\frac{\partial}{\partial z_j} \log p_\mu(\mathbf{Z})\right)\right].
\end{aligned}$$

Stacking the coordinates and using $\ell(\mathbf{s} + \varepsilon\mathbf{Z}) = \mathcal{L}(\mathcal{G}, \Pi(\mathcal{G}, \mathbf{s} + \varepsilon\mathbf{Z}, r))$ gives the score-function identity

$$\nabla_{\mathbf{s}} J_\varepsilon(\mathbf{s}) = \frac{1}{\varepsilon} \mathbb{E}_{\mathbf{Z} \sim \mu}\left[\mathcal{L}\big(\mathcal{G}, \Pi(\mathcal{G}, \mathbf{s} + \varepsilon\mathbf{Z}, r)\big) \xi(\mathbf{Z})\right], \qquad \xi(\mathbf{Z}) = -\nabla_{\mathbf{Z}} \log p_\mu(\mathbf{Z}). \tag{17}$$

□

Relation to the training estimator. A Monte Carlo estimate of Equation (17) would average the sampled loss multiplied by $\xi(\mathbf{Z})/\varepsilon$. Under the assumptions of Proposition 3, this would be an unbiased gradient estimator. The straight-through estimator in Section 3.3 instead uses $\partial\mathcal{L}/\partial\mathbf{P}$ and approximate backward steps through grouping and selection. It is not the score-function estimator above, and the proposition does not establish that its expected update equals $\nabla_{\mathbf{s}} J_\varepsilon(\mathbf{s})$.

D SCORING NETWORK ARCHITECTURE

The scoring network $\psi_{\boldsymbol{\theta}}$ embeds node features, applies graph attention layers, and predicts one score per edge.

Node features. We do not use node attributes. Instead, each node v is described by its local degree profile (Cai & Wang, 2018),

$$\boldsymbol{\ell}_v = \big(d_v,\ \min_{u \in \mathcal{N}(v)} d_u,\ \max_{u \in \mathcal{N}(v)} d_u,\ \mu_v,\ \sigma_v\big), \tag{18}$$

where μ_v, σ_v are the mean and standard deviation of neighbor degrees. We append the first k_{PE} nontrivial Laplacian eigenvectors $\mathbf{p}_v = (u_1(v), \ldots, u_{k_{\mathrm{PE}}}(v))$ (Dwivedi & Bresson, 2020) and use

$$\mathbf{x}_v = \big[\boldsymbol{\ell}_v \,\|\, \mathbf{p}_v\big]. \tag{19}$$

A node MLP with LayerNorm and ReLU maps $\mathbf{x}_v$ to $\mathbf{h}_v^{(0)}$.

Message passing. We use L graph attention layers (Veličković et al., 2018). With $\mathbf{W}^{(l)}$ a linear map and $\mathbf{a}^{(l)}$ the attention vector,

$$c_{uv}^{(l)} = \mathrm{LeakyReLU}\Big(\mathbf{a}^{(l)\top}\big[\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)} \,\|\, \mathbf{W}^{(l)}\mathbf{h}_v^{(l-1)}\big]\Big), \quad \alpha_{uv}^{(l)} = \mathrm{softmax}_u\left(c_{uv}^{(l)}\right), \tag{20}$$

$$\mathbf{h}_v^{(l)} = \mathrm{ReLU}\Big(\textstyle\sum_{u\in\mathcal{N}(v)\cup\{v\}} \alpha_{uv}^{(l)}\,\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}\Big), \tag{21}$$

with H attention heads (concatenated in intermediate layers, averaged in the last). Edge scores are read out from the symmetric embedding of Equation (5) and centered per graph.

E Spectral objectives

Coarse Laplacian. Given the binary coarsening matrix $\mathbf{P}$, the coarse Laplacian is $\widehat{\mathbf{L}} = \mathbf{P}\mathbf{L}\mathbf{P}^\top$; for the normalized Laplacian we follow Cai et al. (2021) and use $\widehat{\mathbf{L}}_{\mathrm{sym}} = \widehat{\mathbf{D}}^{-1/2}\mathbf{P}\mathbf{D}^{1/2}\mathbf{L}_{\mathrm{sym}}\mathbf{D}^{1/2}\mathbf{P}^\top\widehat{\mathbf{D}}^{-1/2}$, where $\mathbf{D}, \widehat{\mathbf{D}}$ are the fine and coarse degree matrices. The unnormalized projection is polynomial in $\mathbf{P}$; the normalized version is differentiable wherever the coarse degrees are positive.

For any coarse signal $\hat{\mathbf{x}}$,

$$\hat{\mathbf{x}}^\top\widehat{\mathbf{L}}\hat{\mathbf{x}} = (\mathbf{P}^\top\hat{\mathbf{x}})^\top\mathbf{L}(\mathbf{P}^\top\hat{\mathbf{x}}),$$

so the coarse quadratic form is induced by the fine Laplacian.

Losses. Let $\mathbf{f}_1, \ldots, \mathbf{f}_k$ be the fine eigenvectors for the k smallest nonzero eigenvalues and $\mathbf{P}^{\mp} = (\mathbf{P}^{+})^\top = \Gamma^{-1}\mathbf{P}$ the map that averages a fine signal within each cluster. We compare the two Laplacians using these fixed vectors:

$$\mathcal{L}_{\mathrm{quad}} = \frac{1}{k}\sum_{i=1}^{k}\left|\mathbf{f}_i^\top\mathbf{L}\mathbf{f}_i - (\mathbf{P}^{\mp}\mathbf{f}_i)^\top\widehat{\mathbf{L}}(\mathbf{P}^{\mp}\mathbf{f}_i)\right|, \tag{22}$$

$$\mathcal{L}_{\mathrm{ray}} = \frac{1}{k}\sum_{i=1}^{k}\left|R_{\mathbf{L}}(\mathbf{f}_i) - R_{\widehat{\mathbf{L}}_{\mathrm{dw}}}\big(\Gamma^{-1/2}\mathbf{P}\mathbf{f}_i\big)\right|, \tag{23}$$

where $R_{\mathbf{A}}(\mathbf{x}) = \mathbf{x}^\top\mathbf{A}\mathbf{x}/\mathbf{x}^\top\mathbf{x}$, $\Gamma = \mathrm{diag}(|\mathcal{C}_1|, \ldots, |\mathcal{C}_{\hat{n}}|)$ holds the cluster sizes, and $\widehat{\mathbf{L}}_{\mathrm{dw}} = \Gamma^{-1/2}\widehat{\mathbf{L}}\Gamma^{-1/2}$ is the doubly weighted coarse Laplacian for which $R_{\widehat{\mathbf{L}}_{\mathrm{dw}}}(\hat{\mathbf{x}}) = R_{\mathbf{L}}(\mathbf{P}^\top\Gamma^{-1/2}\hat{\mathbf{x}})$ (Cai et al., 2021). The quadratic loss compares signal energies; the Rayleigh loss divides each energy by the squared signal norm and is therefore invariant to signal scale. The latter is defined only when the projected signal is nonzero. Normalized Laplacian versions substitute $\mathbf{L}_{\mathrm{sym}}, \widehat{\mathbf{L}}_{\mathrm{sym}}$ and use the degree-adjusted averaging map $\widehat{\mathbf{D}}^{1/2}\mathbf{P}^{\mp}\mathbf{D}^{-1/2}$. The absolute values admit derivatives away from zero and subgradients at zero. The eigenerror in Equation (14) compares the spectra of $\mathbf{L}$ and $\widehat{\mathbf{L}}$ and is used for evaluation and best-of-K inference (evaluation metric), not for training. The doubly weighted operator $\widehat{\mathbf{L}}_{\mathrm{dw}}$ is used only in the Rayleigh loss.

F Straight-through estimator details

The training rule used in the paper is a straight-through estimator. Its forward pass is always the hard greedy coarsening of Algorithm 1, run on $\mathbf{s} + \varepsilon\mathbf{Z}$ with $\mathbf{Z} \sim \mathrm{Gumbel}(0,1)^m$ and $\varepsilon > 0$. The backward pass evaluates the loss gradient at this hard output and supplies surrogate signals through two discrete maps: *grouping* (forest $\to$ partition) and *selection* (scores $\to$ forest). For grouping, we convert the matrix gradient into an edge signal using column norms and normalization. For selection, the base estimator passes this signal through the Gumbel–Softmax top-k relaxation; the optional forest mask replaces this step. Both variants use the same grouping signal.

Edge signals from matrix gradients. We transfer node sensitivity to edges: with $\mathbf{U} = \partial\mathcal{L}/\partial\mathbf{P}$ and $\rho_i = \|\mathbf{U}_{:,i}\|_2$, edge $e = \{i, j\}$ gets $q_e = \rho_i + \rho_j$, normalized to $\tilde{q}_e = q_e/(\frac{1}{m}\sum_{e'} q_{e'} + \delta)$ for a small $\delta > 0$, so $\partial\mathcal{L}/\partial\boldsymbol{\chi} \approx \tilde{\mathbf{q}}$. Taking column norms removes directional information; this is a heuristic grouping signal, not the derivative of the connected-component map.

Gumbel–Softmax top-k relaxation. We construct a relaxed edge-selection vector by iterated softmax with suppression (Plötz & Roth, 2018; Xie & Ermon, 2019): set $\boldsymbol{\alpha}^{(1)} = \mathbf{s} + \varepsilon\mathbf{Z}$ and, for $t = 1, \dots, k$ ($k = n - \hat{n}$), $\mathbf{p}^{(t)} = \mathrm{softmax}(\boldsymbol{\alpha}^{(t)}/\tau)$ and $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + \log(1 - \mathbf{p}^{(t)})$; then $\tilde{\boldsymbol{\chi}} = \sum_{t=1}^{k} \mathbf{p}^{(t)}$ and $\partial\boldsymbol{\chi}/\partial\mathbf{s} \approx \partial\tilde{\boldsymbol{\chi}}/\partial\mathbf{s}$ (Algorithm 2). This relaxation does not enforce acyclicity; the hard forward pass does. With $\varepsilon > 0$, it is the relaxation used by the base STE. Setting $\varepsilon = 0$ removes the perturbation and is used only as an ablation without noise.

Forest mask. The optional forest mask replaces the relaxed selection Jacobian by $\mathrm{diag}(\boldsymbol{\chi})$, restricting the upstream signal to edges in the selected forest:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{s}} \approx \boldsymbol{\chi} \odot \tilde{\mathbf{q}}, \tag{24}$$

This is a coarse gradient (Yin et al., 2019), with zero entries for excluded edges (Algorithm 3). The forward pass and Gumbel perturbation are unchanged.

Algorithm 2: Straight-through backward with Gumbel–Softmax top-k relaxation ($\varepsilon = 0$: ablation without noise)

Input: graph $\mathcal{G}$, scores $\mathbf{s} \in \mathbb{R}^m$, count $k = n - \hat{n}$, temperature $\tau > 0$, noise scale $\varepsilon \geq 0$

sample $\mathbf{Z} \sim \mathrm{Gumbel}(0, 1)^m$; **Forward:** run Algorithm 1 on $\mathcal{G}$ with scores $\mathbf{s} + \varepsilon\mathbf{Z}$ and target $n - k$; return $\mathbf{P}$ and record the selected forest indicator $\boldsymbol{\chi}$

Backward: $\boldsymbol{\alpha}^{(1)} \leftarrow \mathbf{s} + \varepsilon\mathbf{Z}$

for $t = 1$ **to** k **do**

  $\mathbf{p}^{(t)} \leftarrow \mathrm{softmax}(\boldsymbol{\alpha}^{(t)}/\tau)$; **if** $t < k$ **then** $\boldsymbol{\alpha}^{(t+1)} \leftarrow \boldsymbol{\alpha}^{(t)} + \log(1 - \mathbf{p}^{(t)})$

$\tilde{\boldsymbol{\chi}} \leftarrow \sum_{t=1}^{k} \mathbf{p}^{(t)}$; $\partial\boldsymbol{\chi}/\partial\mathbf{s} \approx \partial\tilde{\boldsymbol{\chi}}/\partial\mathbf{s}$

Algorithm 3: Forest-masked straight-through backward pass

Input: graph $\mathcal{G}$, scores $\mathbf{s} \in \mathbb{R}^m$, count $k = n - \hat{n}$, noise scale $\varepsilon > 0$

sample $\mathbf{Z} \sim \mathrm{Gumbel}(0, 1)^m$; **Forward:** run Algorithm 1 on $\mathcal{G}$ with scores $\mathbf{s} + \varepsilon\mathbf{Z}$ and target $n - k$; return $\mathbf{P}$ and record the selected forest indicator $\boldsymbol{\chi}$

Backward: form $\tilde{\mathbf{q}}$ from $\partial\mathcal{L}/\partial\mathbf{P}$ using Equation (10); set $\partial\mathcal{L}/\partial\mathbf{s} \leftarrow \boldsymbol{\chi} \odot \tilde{\mathbf{q}}$

G Computational complexity

We separate the cost of the coarsening operator from objective evaluation and differentiation. Assuming node features are available, with the fixed architecture used in all experiments, the scoring network processes every node and edge a constant number of times and therefore costs $O(n + m)$. The symmetric edge readout and score centering add $O(m)$ operations.

The hard operator first sorts the m edge scores in $O(m \log m)$ time. The union–find pass costs $O(m\alpha(n))$, and converting its connected components into cluster assignments costs $O(n)$. Deterministic inference therefore costs

$$O(n + m) + O(m \log m) + O(m\alpha(n) + n) = O(n + m \log m).$$

This bound assumes that the partition is stored as cluster labels or as a sparse coarsening matrix, which has exactly n nonzero entries. Explicitly materializing a dense $\hat{n} \times n$ matrix instead requires $O(n\hat{n})$ memory and initialization time.

The hard forward pass used during training has the same complexity. Given the upstream matrix gradient $\mathbf{U} = \partial\mathcal{L}/\partial\mathbf{P}$, computing its node norms and projecting them to the edges costs $O(n\hat{n} + m)$. The base straight-through estimator then performs $k = n - \hat{n}$ relaxed selections over all m edges, which costs $O(km)$ in the backward pass. The forest mask replaces this relaxation by an $O(m)$ operation. These bounds describe only the operator-side training cost and exclude objective evaluation and differentiation.

For sampled inference, we compute the edge scores once. Generating K perturbed candidates and one deterministic candidate costs

$$O\big(n + m + (K+1)(n + m\log m)\big).$$

If $C_{\mathcal{L}}$ is the cost of evaluating the selection objective, selecting the best candidate additionally requires $(K+1)C_{\mathcal{L}}$. Figure 14 compares the inference time of deterministic and best-of-8 inference (evaluation metric).

Finally, the Laplacian positional encodings are a separate feature-preprocessing cost and are computed once per graph. The current implementation forms a dense Laplacian and computes its full eigendecomposition before retaining the first k_{PE} nontrivial eigenvectors, requiring $O(n^3)$ time and $O(n^2)$ memory. A partial sparse eigensolver can instead compute the required $k_{\mathrm{PE}} + 1$ smallest eigenvectors. If q Krylov vectors are needed, this requires $O(qm + nq^2)$ time and $O(m + nq)$ memory. When $q = O(k_{\mathrm{PE}})$, this becomes $O(k_{\mathrm{PE}}m + nk_{\mathrm{PE}}^2)$ time and $O(m + nk_{\mathrm{PE}})$ memory. These preprocessing costs are not included in the core operator bounds above.

H Effect of score perturbations and the forest mask

Figure 5 compares three variants of the training rule on PROTEINS. The left column removes the perturbation ($\varepsilon = 0$) but keeps the hard greedy forward pass and top-k backward relaxation. The resulting score-to-loss map, $J_0(\mathbf{s}) = \mathcal{L}(\mathcal{G}, \Pi(\mathcal{G}, \mathbf{s}, r))$, is piecewise constant. Without perturbation, the displayed runs become unstable at $r \in \{0.25, 0.75\}$.

The center column adds Gumbel perturbations. These define the smoothed expected loss in Proposition 3; training uses the heuristic backward rule described in Section F. Perturbation reduces the instability in these runs, but substantial oscillations remain at $r = 0.75$. The right column adds the forest mask from Equation (13), giving steadier curves at this ratio.

I Learned loss model

In this alternative, we keep the discrete operator unchanged and train a differentiable network $\widehat{\mathcal{L}}_{\boldsymbol{\eta}}(\mathcal{G}, \mathbf{s}, r)$ to predict $\mathcal{L}(\mathcal{G}, \Pi(\mathcal{G}, \mathbf{s}, r))$ from the edge scores. We fit the loss network to evaluations of the true coarsening loss and update the scoring network by minimizing $\widehat{\mathcal{L}}_{\boldsymbol{\eta}}(\mathcal{G}, \psi_{\boldsymbol{\theta}}(\mathcal{G}), r)$. Updates to $\boldsymbol{\theta}$ change the score distribution on which the loss network is trained, so we fit the two networks alternately.

Architecture. We embed node features $\mathbf{x}_v$ and edge scores s_e with separate MLPs and condition these embeddings on the reduction ratio through FiLM (Perez et al., 2018): $\mathrm{FiLM}(\mathbf{z}, r) = \mathbf{z} \odot (\mathbf{1} + \boldsymbol{\gamma}(r)) + \boldsymbol{\beta}(r)$, with $\boldsymbol{\gamma}, \boldsymbol{\beta}$ from a small SiLU network of r. Message passing uses attention that includes edge features in the logits, a residual with learned scale, LayerNorm, and SiLU, with FiLM injected after each layer. A Jumping Knowledge readout (Xu et al., 2018) concatenates mean and max pooled states from all layers (Corso et al., 2020), is conditioned by FiLM once more, and an MLP with a softplus output produces the nonnegative predicted loss.

Training. We first pretrain the loss network for T_{pre} epochs, keeping the scoring network fixed and fitting the true loss with a Huber objective (Huber, 1964). We then alternate K_{loss} loss network updates with K_{score} scoring network updates (Algorithm 4). We update the scorer by differentiating the *predicted* loss, never the true loss.

J Results with the learned loss model

The best schedule we found (K_{loss}:$K_{\mathrm{score}} = 1$:2, $T_{\mathrm{pre}} \approx 5$) did not reliably reduce the true coarsening loss. Figure 6 shows that the loss network's predictions can track the true loss in some runs, but the fit varies across reduction ratios and training stages. During joint training, reducing the predicted loss does not consistently reduce the true loss. These results suggest that prediction accuracy alone is insufficient to obtain useful gradients with respect to the scores; they do not isolate whether the failure comes from approximation error, distribution shift, or the learned gradients.

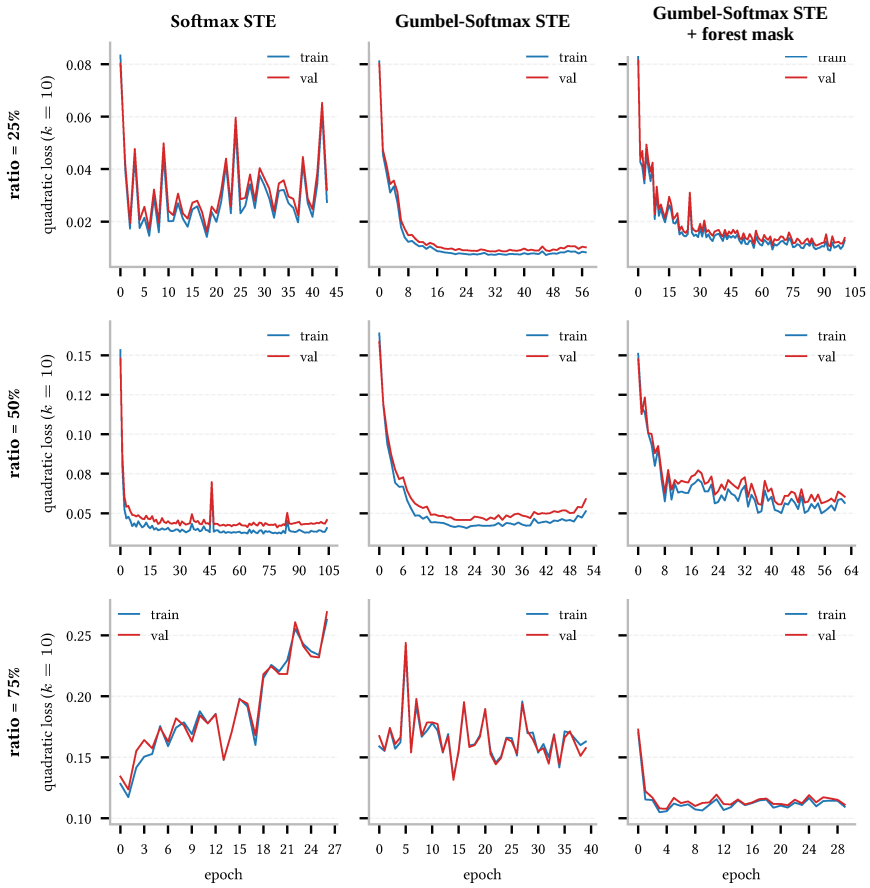


Figure 5: Effect of score perturbations and the forest mask on PROTEINS with the quadratic loss; rows are reduction ratios. "Softmax STE" removes perturbation, "Gumbel–Softmax STE" is the base estimator, and "Gumbel–Softmax STE + forest mask" restricts the backward signal to the selected forest. Perturbation reduces instability, and the forest mask gives steadier curves at high reduction.

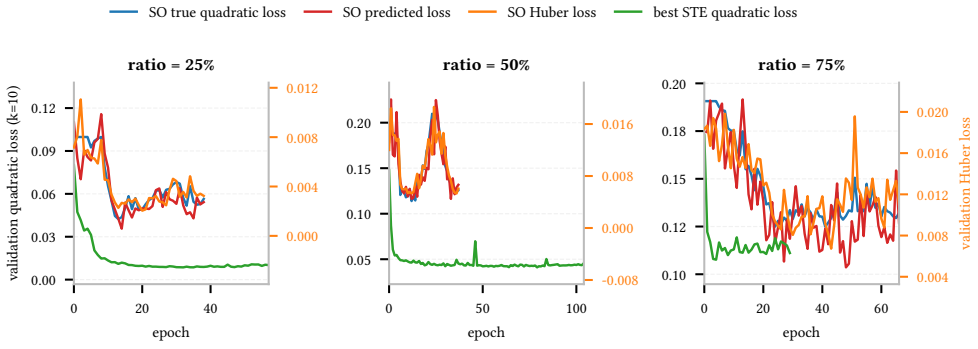


Figure 6: Predicted and true coarsening loss for the surrogate objective approach. The fit varies across runs, and joint training does not reliably reduce the true loss.

Algorithm 4: Training with a learned loss model

Input: data $\mathcal{D}$, params $\boldsymbol{\theta}, \boldsymbol{\eta}$, pretrain epochs T_{pre}, epochs T, schedule $K_{\text{loss}}, K_{\text{score}}$

for $t = 1$ **to** T_{pre} **do**
  foreach *batch* $\mathcal{B}$ **do**
    freeze $\boldsymbol{\theta}$; $\mathbf{s} \leftarrow \psi_{\boldsymbol{\theta}}(\mathcal{B})$; $y \leftarrow \mathcal{L}(\mathcal{B}, \Pi(\mathcal{B}, \mathbf{s}, r))$; $\hat{y} \leftarrow \widehat{\mathcal{L}}_{\boldsymbol{\eta}}(\mathcal{B}, \mathbf{s}, r)$; update $\boldsymbol{\eta}$ on $\mathrm{Huber}(\hat{y}, y)$
for $t = 1$ **to** T **do**
  $c \leftarrow 0$
  foreach *batch* $\mathcal{B}$ **do**
    if $c < K_{\text{loss}}$ **then**
      freeze $\boldsymbol{\theta}$; $\mathbf{s} \leftarrow \psi_{\boldsymbol{\theta}}(\mathcal{B})$; update $\boldsymbol{\eta}$ on $\mathrm{Huber}(\widehat{\mathcal{L}}_{\boldsymbol{\eta}}(\mathcal{B}, \mathbf{s}, r), \mathcal{L}(\mathcal{B}, \Pi(\mathcal{B}, \mathbf{s}, r)))$
    else
      freeze $\boldsymbol{\eta}$; update $\boldsymbol{\theta}$ on $\widehat{\mathcal{L}}_{\boldsymbol{\eta}}(\mathcal{B}, \psi_{\boldsymbol{\theta}}(\mathcal{B}), r)$
    $c \leftarrow (c + 1) \bmod (K_{\text{loss}} + K_{\text{score}})$

K Experimental setup

Datasets. Table 1 summarizes the test split of the final benchmark. All graphs are made undirected, unweighted, connected, and free of self loops before coarsening. The benchmark uses fixed train/validation/test splits for all methods. Metric eligibility is determined by the prescribed target size. Computing ten nonzero eigenvalues requires at least 11 coarse nodes. Our forest construction reaches the target size exactly, whereas some classical methods retain more nodes than requested and can therefore yield an eigenerror even when the prescribed target has at most ten nodes. To ensure paired comparisons, every method is evaluated on the common subset satisfying $\hat{n}(r) > k$. At $r = 0.75$, this leaves 15 PROTEINS, 6 ENZYMES, and 32 COLLAB test graphs. A CORA run produced valid training graphs but no validation or test graphs under this fixed sampling protocol, so it is excluded from the aggregate test tables rather than filled with imputed values. For the Erdős–Rényi family, enforcing the connected-graph assumption leaves 24 graphs across five expected-degree regimes. The fixed split assigns 17 graphs to training, 4 to validation, and 3 to testing.

Table 1: Test set graph statistics for the final benchmark. The benchmark includes molecular and collaboration graphs from TU benchmarks (Morris et al., 2020), PUBMED citation subgraphs (Namata et al., 2012; Yang et al., 2016), Amazon copurchase subgraphs (Shchur et al., 2019), and synthetic SBM, Erdos–Renyi, and Barabasi–Albert graphs (Holland et al., 1983; Erdős & Rényi, 1959; Barabási & Albert, 1999).

Dataset	Test graphs	Mean n	Mean m	Mean degree	Degree Gini
PROTEINS	45	39.3	74.4	3.8	0.13
ENZYMES	45	32.0	64.7	4.0	0.12
COLLAB	45	68.2	1664.8	32.8	0.23
PUBMED	45	250.7	298.2	2.4	0.41
AMAZON PHOTO	45	271.4	817.9	5.9	0.52
AMAZON COMPUTERS	45	282.0	738.9	5.1	0.48
SBM	45	186.0	768.4	7.8	0.20
ER	3	182.3	1238.7	13.0	0.20
BA	45	189.4	1173.7	12.7	0.32

Baselines and metric. Classical baselines use the implementations of Loukas (2019). The main classical comparison selects, on validation, the best method among Heavy Edge (Karypis & Kumar, 1998), Algebraic JC (Ron et al., 2011), Affinity GS (Livne & Brandt, 2012), Kron reduction (Dörfler & Bullo, 2013), Variation Edges (Loukas, 2019), and Variation Neighborhoods (Loukas, 2019). Random edge contraction is reported separately. Heavy Edge and Affinity GS are included in the

validation pool but are not selected in any final cell. Quality is measured by the relative eigenerror in Equation (14), comparing the $k = 10$ smallest nonzero eigenvalues of the fine Laplacian and the size-normalized coarse Laplacian.

Model and training settings. The core scoring network uses 5 attention layers, hidden width 64, 4 heads, LDP and positional encoding inputs, and an edge MLP of widths $(64, 32, 16)$; training uses Adam at learning rate 10^{-4}, batch size 3, up to 300 epochs with early stopping (patience 25). The relaxation temperature is annealed from $\tau = 10$ by $\times 0.95$ every 5 epochs; the perturbation scale of the straight-through estimator is $\varepsilon = 1$ (both chosen in Section Q). The surrogate objective model uses the schedule of Section I. The final benchmark uses three training seeds (42, 43, and 44); reported means aggregate the evaluated graphs and runs.

L Results by dataset and reduction ratio

Table 2: **Comparison with classical methods.** "Wins vs. classical" counts datasets where best-of-K inference (evaluation metric) has lower test eigenerror than the classical baseline selected on validation. Gains are relative error reductions.

r	Datasets	Wins vs. classical	Mean gain	Median gain
0.25	9	7/9	12.8%	18.1%
0.50	9	8/9	37.0%	41.4%
0.75	9	9/9	61.3%	73.2%

Table 3 gives the full results behind Figure 2 and the aggregate values in Table 2. The columns separate the averaged Random baseline, the stronger Random best of 128 baseline, the deterministic inference output, best-of-K inference (training loss), best-of-K inference (evaluation metric), and *Gumbel–Softmax STE + forest mask* at $r = 0.75$. Random best of 128 draws 128 random contraction forests per test graph and keeps the one with the lowest eigenerror. The "Best" column identifies the method with the lowest mean test eigenerror in each cell; it is not a validation selection rule.

Table 3: Results by dataset and reduction ratio. Each cell lists the mean eigenerrors for $r = 0.25, 0.50, 0.75$ from top to bottom; lower is better. Random best of 128 evaluates 128 random contraction forests using the same objective (eigenerror) as best-of-K inference (evaluation metric). Deterministic inference uses the learned scores with $\varepsilon = 0$. Both learned sampling modes include the deterministic forest: best-of-K inference (training loss) minimizes the training loss, and best-of-K inference (evaluation metric) minimizes eigenerror. The forest-masked variant is evaluated only for $r = 0.75$. In the "Best" column, C, R, E, and M denote the best classical baseline, Random best of 128, evaluation selection, and forest mask, respectively. All methods use the common eligible graph subset defined in Section K. GOREN is compared separately in Tables 4 and 5.

Dataset	Random	Random best 128	Best classical	Det.	Training selection	Evaluation selection	Forest mask	Best
	0.520	0.349	0.330	0.427	0.387	0.327	–	E
	1.684	1.210	1.236	1.466	1.367	1.147	–	E
PROTEINS	5.685	3.996	4.955	4.728	4.804	3.935	3.662	M
	0.482	0.328	0.312	0.420	0.378	0.312	–	E
	1.521	1.058	1.159	1.331	1.247	1.026	–	E
ENZYMES	4.945	3.303	4.329	3.523	3.865	3.081	2.950	M
	0.157	0.046	0.075	0.049	0.051	0.039	–	E
	0.357	0.133	1.605	0.159	0.151	0.111	–	E
COLLAB	0.838	0.371	2.303	0.577	0.467	0.326	0.335	E
	0.619	0.365	0.242	0.219	0.210	0.177	–	E
	2.000	1.238	1.243	0.778	0.755	0.656	–	E
PUBMED	7.256	4.118	4.608	3.721	3.428	2.908	3.247	E
	0.231	0.088	0.037	0.030	0.029	0.025	–	E
	0.680	0.332	0.255	0.130	0.123	0.105	–	E
AMAZON PHOTO	1.942	1.076	2.217	0.745	0.711	0.593	0.523	M
	0.288	0.116	0.058	0.050	0.047	0.043	–	E
	0.793	0.413	0.346	0.175	0.164	0.145	–	E
AMAZON COMPUTERS	2.186	1.203	2.429	0.661	0.619	0.537	0.595	E
	0.198	0.109	0.065	0.116	0.098	0.084	–	C
	0.443	0.262	0.163	0.193	0.185	0.169	–	C
SBM	0.816	0.491	1.782	0.371	0.364	0.335	0.365	E
	0.189	0.059	0.070	0.068	0.061	0.057	–	E
	0.461	0.135	0.149	0.133	0.133	0.120	–	E
ER	0.796	0.257	1.906	0.356	0.330	0.288	0.344	R
	0.141	0.099	0.077	0.096	0.096	0.083	–	C
	0.308	0.232	0.252	0.168	0.166	0.148	–	E
BA	0.550	0.431	0.760	0.331	0.330	0.290	0.320	E

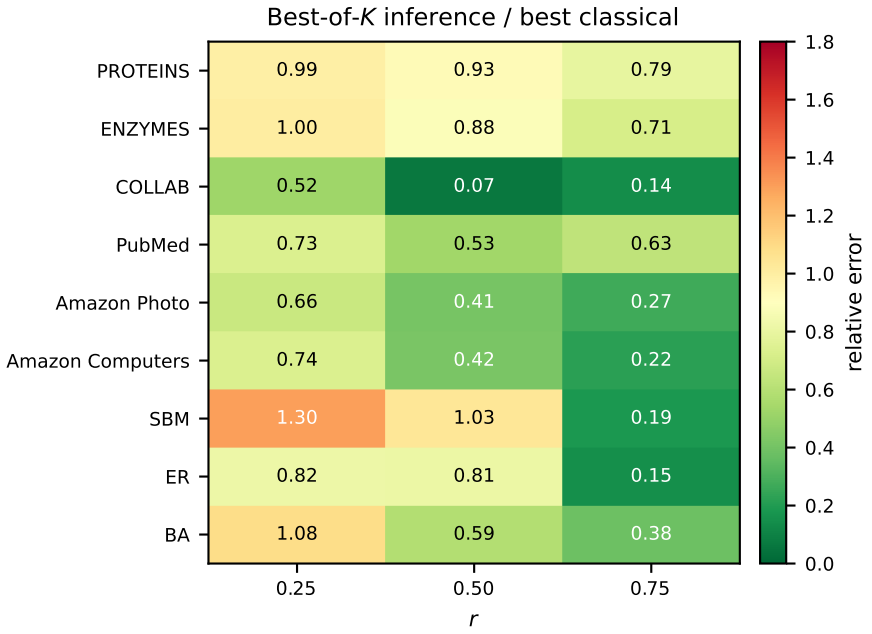


Figure 7: Full relative error matrix for the final benchmark. Values are learned eigenerror under best-of-K inference (evaluation metric) divided by the classical eigenerror selected on validation. Values below one indicate that learned coarsening improves over the best classical baseline.

The aggregation starts from 19629 baseline result rows and 6263460 learned method result rows before validation selection and aggregation. Among the methods in Table 3, the learned operator with best-of-K inference (evaluation metric) has the lowest mean eigenerror in 20 of the 27 cells, *Gumbel–Softmax STE + forest mask* is best in 3 cells, and the classical baseline is best in 3 cells. Random best of 128 is best in 1 cell.

M Comparison with GOREN

Matched quadratic objective. Table 4 reports the complete comparison when both methods are trained with the quadratic loss. Deterministic inference gives lower test eigenerror than GOREN in 16 of 27 settings, including six of nine datasets at both $r = 0.50$ and 0.75.

Table 4: **Comparison with GOREN using the quadratic training loss.** Relative test eigenerror reduction (%) compared with GOREN, using deterministic inference. Positive values are bold and indicate lower error for our operator. GOREN's classical coarsening method is selected on validation data.

Dataset	$r = 0.25$	$r = 0.50$	$r = 0.75$
PROTEINS	-78.9	-51.1	-73.7
ENZYMES	-75.6	-42.2	-126.2
COLLAB	**40.3**	**89.6**	**73.2**
PUBMED	**5.0**	**25.3**	-6.0
AMAZON PHOTO	**19.8**	**33.3**	**52.1**
AMAZON COMPUTERS	**5.6**	**38.3**	**64.5**
SBM	-144.6	-61.6	**76.3**
ER	-1.0	**12.4**	**83.7**
BA	-34.8	**29.7**	**72.9**
Lower eigenerror	4/9	6/9	6/9

Original Rayleigh objective. Table 5 compares our operator with GOREN trained using its original Rayleigh objective. Our operator remains trained with the quadratic loss. Deterministic inference gives lower test eigenerror than GOREN in 17 of 27 settings. The win/loss pattern differs from the quadratic-loss comparison in Table 4 only on ER at $r = 0.25$, where our operator also has lower error. GOREN remains better on PROTEINS and ENZYMES at all three reduction ratios.

Table 5: Relative test eigenerror reduction (%) compared with GOREN, using deterministic inference. GOREN is trained with the Rayleigh objective and our operator with the quadratic loss. Positive values are bold and indicate lower error for our operator.

Dataset	$r = 0.25$	$r = 0.50$	$r = 0.75$
PROTEINS	-80.9	-56.5	-112.7
ENZYMES	-71.0	-56.5	-46.6
COLLAB	**37.2**	**90.9**	**73.9**
PUBMED	**5.0**	**25.3**	-6.0
AMAZON PHOTO	**16.1**	**33.1**	**52.1**
AMAZON COMPUTERS	**5.4**	**38.3**	**64.6**
SBM	-145.5	-59.0	**77.1**
ER	**9.0**	**8.1**	**83.8**
BA	-15.4	**29.4**	**74.0**
Lower eigenerror	5/9	6/9	6/9

N COMPARISON OF INFERENCE MODES

Table 6 and Figure 8 compare the two sampled inference rules with deterministic inference. Best-of-K inference (training loss) avoids computing coarse eigenvalues for candidate selection and improves the deterministic result in most cells. Best-of-K inference (evaluation metric) selects by eigenerror and improves the deterministic result in every cell, at the cost of an eigendecomposition for each candidate. The number of candidates is selected on validation from $K \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ together with the perturbation scale. For the base STE, best-of-K inference (evaluation metric) chooses $K = 128$ in every final cell; best-of-K inference (training loss) may choose a smaller K because it evaluates candidates with the training loss rather than the evaluation objective. The main text comparison also includes Random best of 128, which applies the same selection rule based on the objective to random contraction forests instead of learned score perturbations.

Table 6: Inference mode summary across datasets. Values are relative to the output of deterministic inference. The win rate is the fraction of datasets where the sampled mode has lower eigenerror than deterministic inference.

r	Mode	Median rel.	Q1	Q3	Win rate
0.25	Best-of-K (training loss)	0.948	0.904	0.982	89%
0.25	Best-of-K (evaluation metric)	0.809	0.765	0.840	100%
0.50	Best-of-K (training loss)	0.946	0.939	0.970	100%
0.50	Best-of-K (evaluation metric)	0.831	0.782	0.873	100%
0.75	Best-of-K (training loss)	0.955	0.928	0.998	78%
0.75	Best-of-K (evaluation metric)	0.813	0.796	0.874	100%

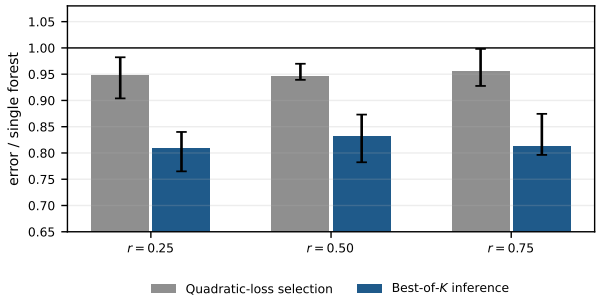


Figure 8: Median test error of sampled inference modes relative to deterministic inference. Best-of-K inference (training loss) uses K selected on validation; best-of-K inference (evaluation metric) uses $K = 128$ perturbed forests plus the deterministic forest for the base STE. Error bars show the interquartile range across datasets.

O Effect of the forest mask at high reduction

Table 7 and Figure 10 compare the Gumbel–Softmax STE and *Gumbel–Softmax STE + forest mask* at $r = 0.75$. The forest mask improves the mean eigenerror on 3 of nine datasets; the base estimator is at least as good on the others.

Figure 9 shows the result by dataset. The forest-masked variant has the lowest mean eigenerror among the methods in Table 3 on 3 of nine datasets, namely PROTEINS, ENZYMES, and AMAZON PHOTO.

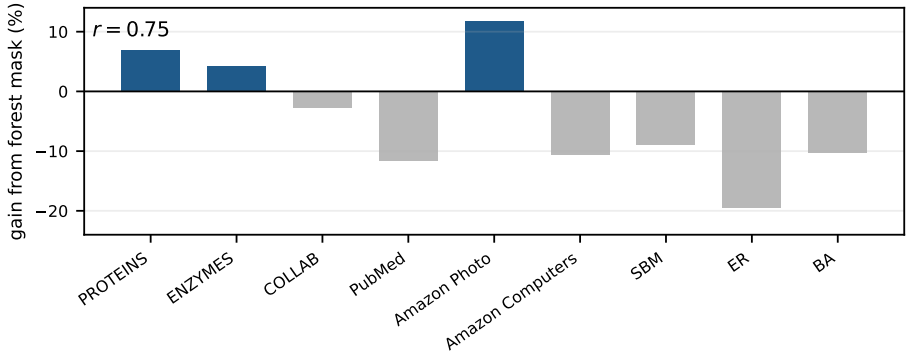


Figure 9: **Effect of the forest mask at high reduction.** Bars show the relative eigenerror reduction of *Gumbel–Softmax STE + forest mask* over the base best-of-K inference (evaluation metric) mode at $r = 0.75$. Positive values mean that the forest mask improves the base method.

P Comparison of training losses

Table 8 compares quadratic and Rayleigh training losses by validation eigenerror. The results are mixed: the quadratic loss performs better on COLLAB, while the Rayleigh loss performs better on PROTEINS and SBM. CORA has no finite validation result. The final benchmark uses the quadratic loss for all learned models.

Table 7: Comparison at high reduction for $r = 0.75$. "Mask gain" is the relative eigenerror reduction of *Gumbel–Softmax STE + forest mask* over the base best-of-K inference (evaluation metric) mode, so positive values mean that the forest mask helps.

Dataset	Best classical	Gumbel–Softmax STE	Gumbel–Softmax STE + forest mask	Mask gain
PROTEINS	4.955	3.935	3.662	6.9%
ENZYMES	4.329	3.081	2.950	4.2%
COLLAB	2.303	0.326	0.335	-2.7%
PUBMED	4.608	2.908	3.247	-11.7%
AMAZON PHOTO	2.217	0.593	0.523	11.8%
AMAZON COMPUTERS	2.429	0.537	0.595	-10.7%
SBM	1.782	0.335	0.365	-9.0%
ER	1.906	0.288	0.344	-19.5%
BA	0.760	0.290	0.320	-10.3%

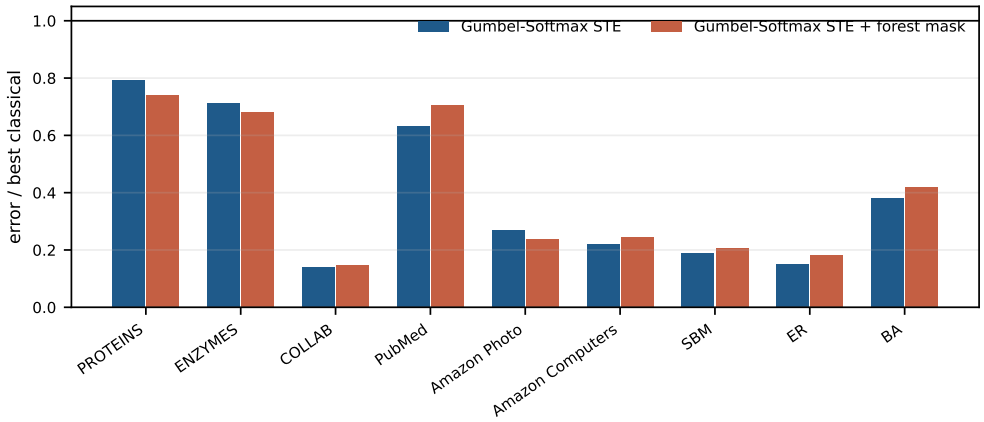


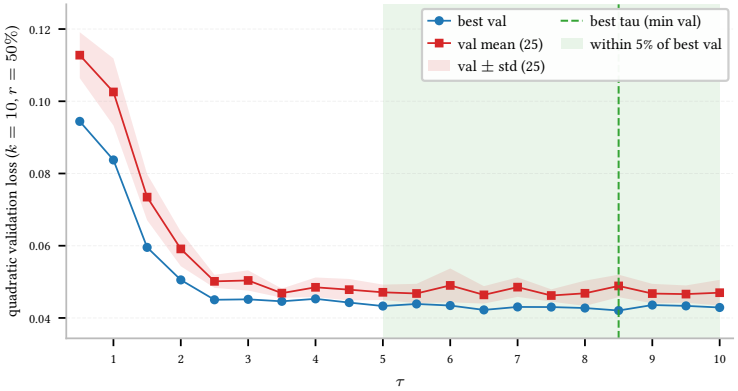
Figure 10: Comparison at high reduction for $r = 0.75$. Both learned variants are below the classical baseline on every dataset, while the forest mask is beneficial only on a subset of graph families.

Table 8: Validation eigenerror with quadratic and Rayleigh training losses. Cora has no finite validation score under the fixed split and is therefore excluded from the final aggregate benchmark.

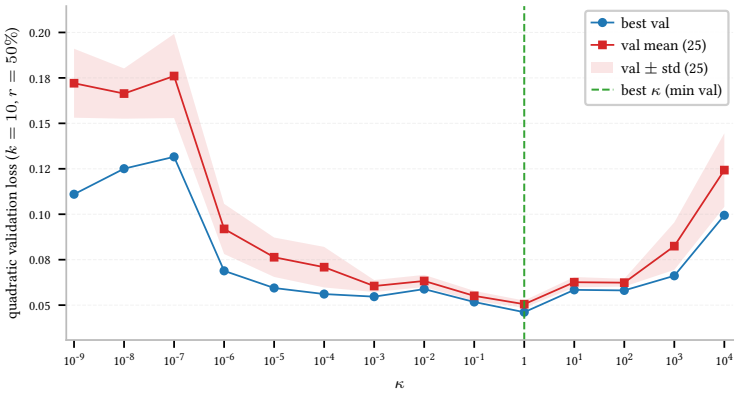
Dataset	Loss	Val. error	Std.	Runs
COLLAB	quadratic	0.124	0.001	3
COLLAB	rayleigh	0.135	0.005	3
CORA	quadratic	–	–	3
CORA	rayleigh	–	–	3
PROTEINS	quadratic	1.435	0.042	3
PROTEINS	rayleigh	1.304	0.010	3
SBM	quadratic	0.171	0.010	3
SBM	rayleigh	0.142	0.002	3

Q TEMPERATURE AND NOISE SCALE

In Figure 11, temperatures $\tau \in [5, 10]$ give losses within 5% of the best value; we anneal from $\tau = 10$. Among the tested perturbation scales, $\varepsilon = 1$ (shown as κ in the figure) performs best. Smaller scales change the ranking less often, while larger scales give the noise more influence over edge selection.



(a) temperature τ

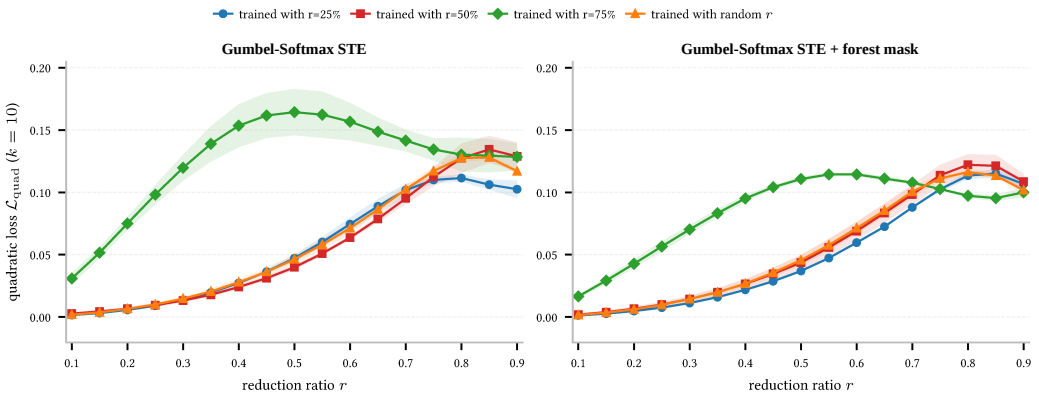


(b) noise scale κ

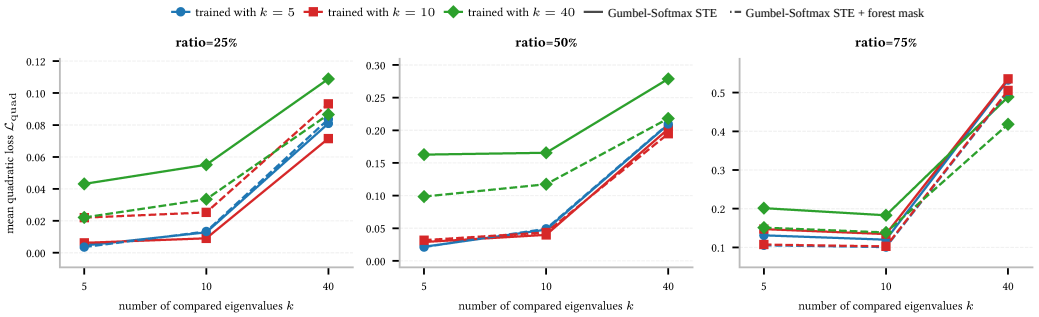
Figure 11: Effect of temperature and noise scale on PROTEINS ($k = 10$, $r = 0.5$).

R TRANSFER ACROSS GRAPHS AND COARSENING SETTINGS

Figure 12 evaluates transfer using the quadratic loss. Training at small or moderate reduction ratios and with $k \in \{5, 10\}$ transfers well across the tested ratios and eigenvector counts. The graph-size trend depends on the reduction ratio and is not monotone at high reduction. A model trained on PROTEINS also transfers better to ENZYMES than to COLLAB; these experiments do not establish which graph properties explain the difference.

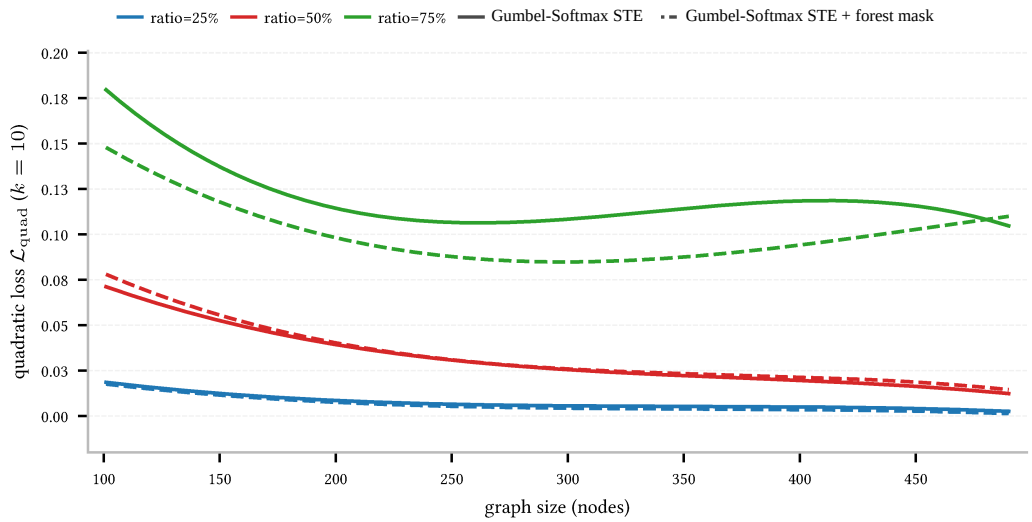


(a) over r

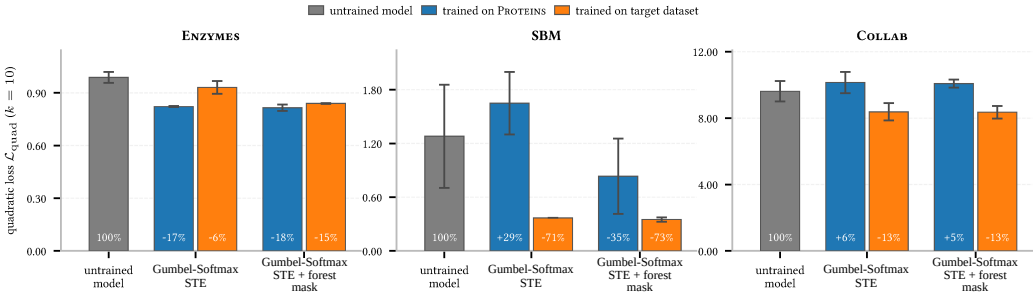


(b) over k

Figure 12: Quadratic loss under changes in reduction ratio, eigenvector count, graph size, and dataset. The default is $k = 10$, except in the eigenvector-count study. Lower is better. Panels (c) and (d) continue on the next page.



(c) over size



(d) across datasets

Figure 12: Quadratic loss under changes in graph size and dataset (continued). Lower is better.

S RESULTS ON LARGER GRAPHS

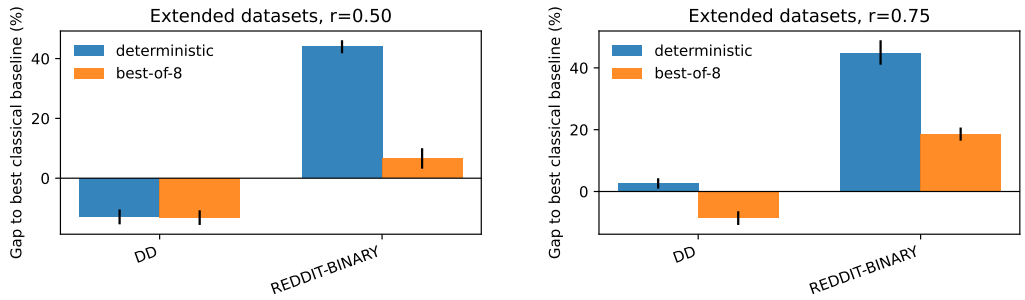
The final benchmark is complemented by a larger graph stress test on DD and REDDIT-BINARY. These runs use the same model code and training settings, but limit evaluation to the test subsets in Table 9. These experiments test the learned operator on larger graphs.

Table 9: Test set graph statistics for the larger graph stress test.

Dataset	Test graphs	Mean n	Mean m	Min n	Max n
DD	105	278.4	703.2	37	640
REDDIT-BINARY	97	184.8	221.1	10	437

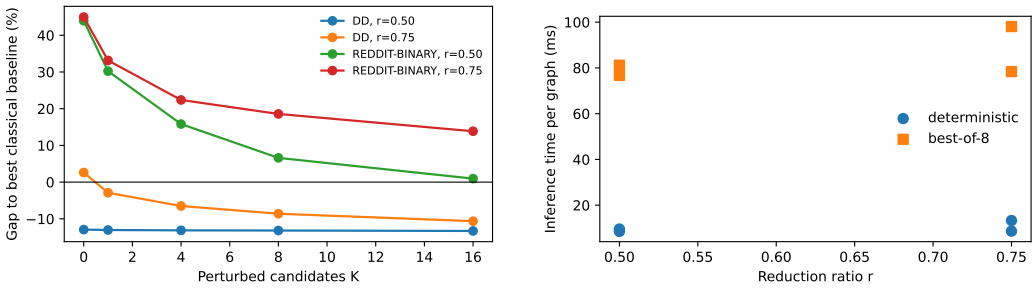
Table 10: Stress test gap to the best classical method (%, mean±std across three runs). Negative values mean that learned coarsening has lower eigenerror than the classical baseline selected on validation.

Dataset	Mode	$r = 0.5$	$r = 0.75$
DD	Deterministic	-12.9 ± 2.5	2.6 ± 1.7
DD	Best-of-8 (evaluation metric)	-13.2 ± 2.5	-8.6 ± 2.2
REDDIT-BINARY	Deterministic	43.9 ± 2.2	45.0 ± 4.0
REDDIT-BINARY	Best-of-8 (evaluation metric)	6.6 ± 3.4	18.6 ± 2.1



(a) $r = 0.50$ (b) $r = 0.75$

Figure 13: Stress test gap to the classical baseline selected on validation. Sampling improves the learned results, but best-of-8 inference (evaluation metric) does not close the gap on REDDIT-BINARY. Negative values favor learned coarsening.

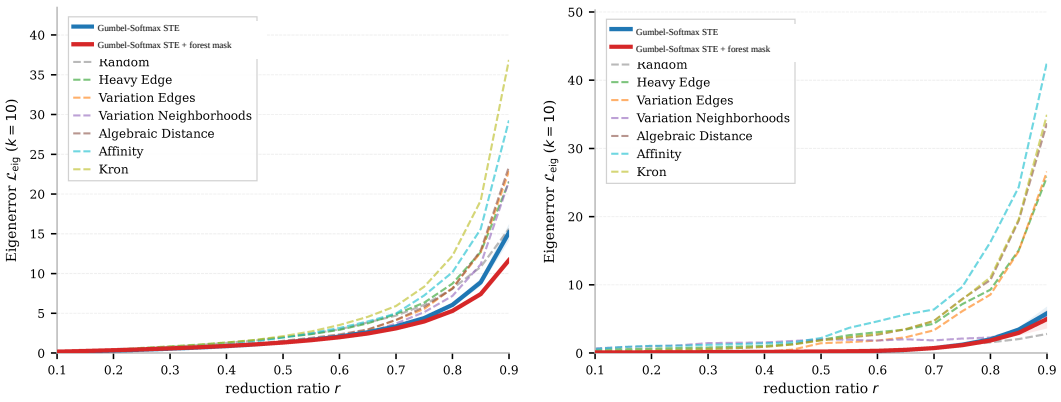


(a) best-of-K inference (evaluation metric) (b) inference time

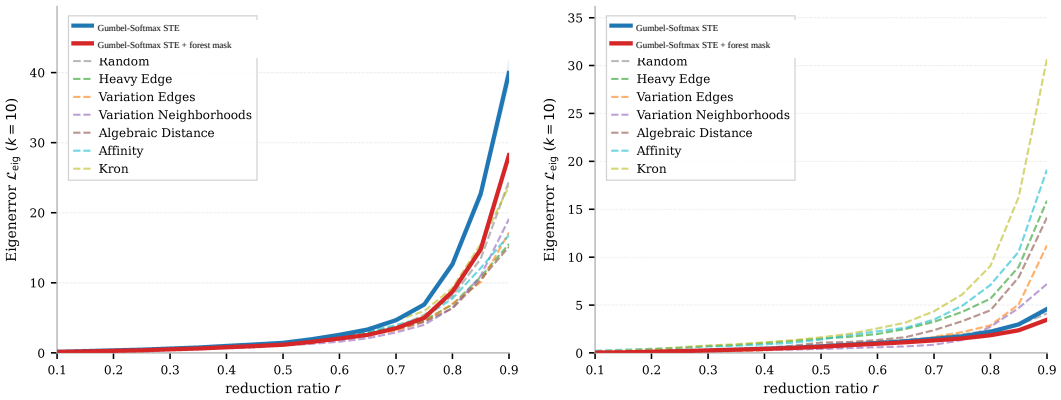
Figure 14: Candidate budget and inference cost on the stress test datasets. Left: eigenerror gap as the candidate budget increases. Right: inference time for deterministic and best-of-8 inference (evaluation metric).

T UNNORMALIZED SPECTRUM

Figure 15 reports a supplementary evaluation over a finer grid of reduction ratios on four datasets. Models are trained with the quadratic loss and evaluated by eigenerror with $k = 10$.



(a) Proteins (b) Collab

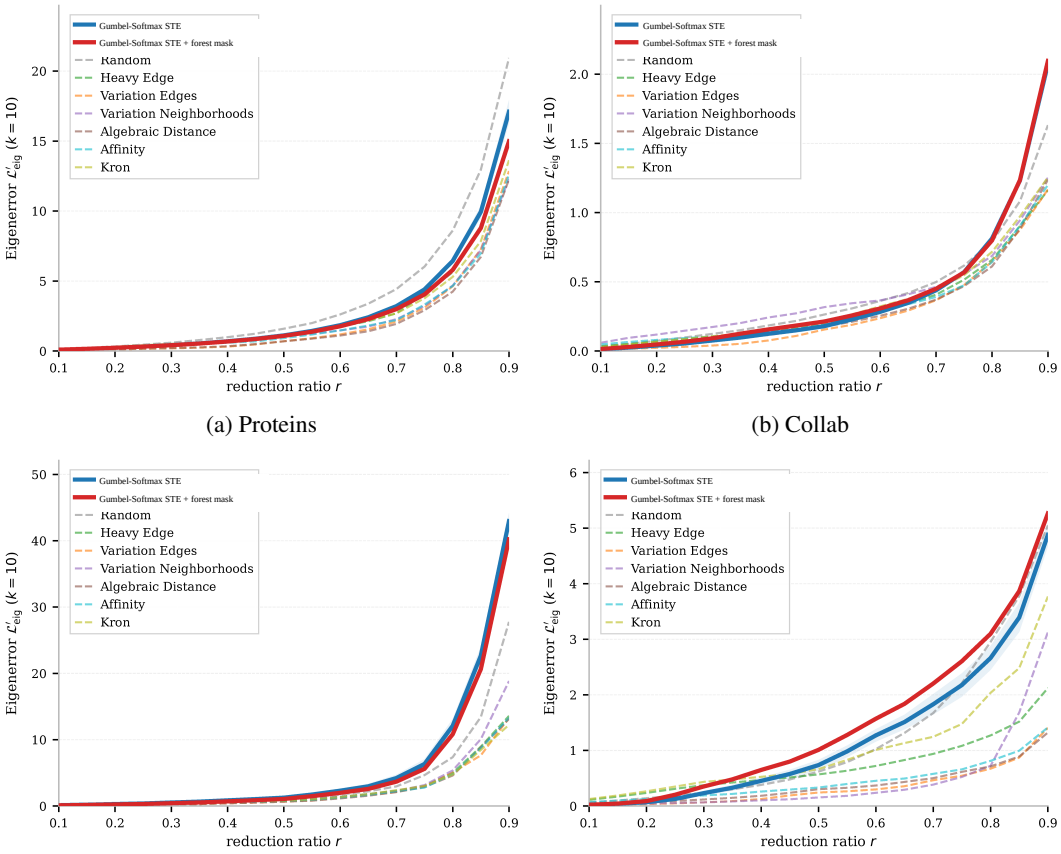


(c) Enzymes (d) SBM

Figure 15: Unnormalized Laplacian eigenerror vs. r ($k = 10$). Lower is better. Learned methods are shown as thicker solid lines, while random and classical baselines are shown as thinner dashed lines.

U NORMALIZED SPECTRUM

Figure 16 evaluates preservation of the normalized Laplacian spectrum. The learned methods' advantage is less consistent than in the unnormalized evaluation and varies with the dataset and reduction ratio. Degree normalization changes the effect of a contraction on the Laplacian, but these experiments do not isolate the cause of the performance differences.



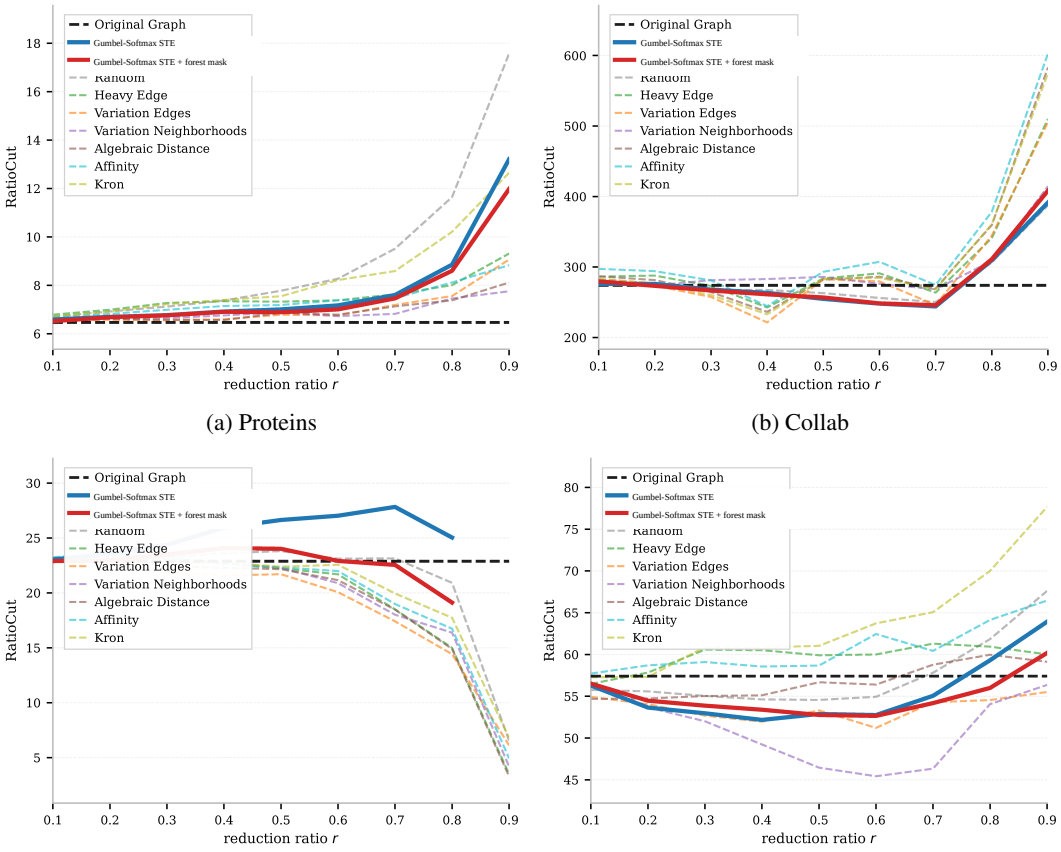
(a) Proteins (b) Collab

(c) Enzymes (d) SBM

Figure 16: Normalized Laplacian eigenerror vs. r ($k = 10$). Learned methods are shown as thicker solid lines, while random and classical baselines are shown as thinner dashed lines.

V Downstream spectral clustering

We run spectral clustering with $c = 10$ clusters on each coarse graph, assign its labels to the original nodes through $\mathbf{P}$, and evaluate the resulting partition by RatioCut. Figure 17 compares methods over reduction ratios from 0.1 to 0.9, with clustering on the original graph as a reference. The learned operators are competitive on PROTEINS and COLLAB but do not consistently outperform the baselines, particularly on SBM. The quadratic training loss and eigenerror evaluation do not directly optimize RatioCut; training with a clustering objective is a possible next step.



(a) Proteins

(b) Collab

(c) Enzymes

(d) SBM

Figure 17: Downstream spectral clustering: RatioCut vs. r ($c = 10$), lower is better. Learned methods are shown as thicker solid lines, while random and classical baselines are shown as thinner dashed lines.