Variational Graph Structure Learning for GNNs by using Marginal Likelihood

Anonymous authors
Paper under double-blind review

Abstract

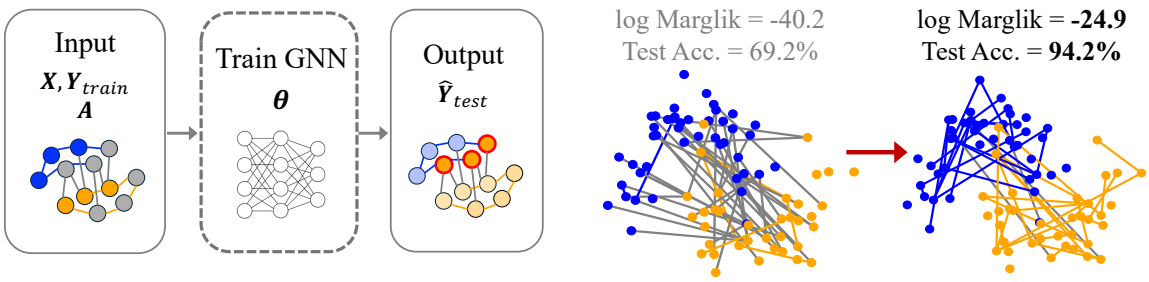
Learning graph structures for Graph Neural Networks (GNNs) can improve their performance, but it is challenging to search over the large discrete space of graphs. Prior works often impose fixed structural constraints to promote properties such as sparsity, but these constraints can be misspecified and overly restrictive, potentially degrading performance. Here, we propose a simpler alternative based on marginal likelihood, which naturally favors such properties without requiring any explicit graph constraints. We show that a variational formulation with Laplace's method automatically leads to a marginal-likelihood based objective over discrete graph structures, which can be optimized efficiently using the Gumbel-Softmax trick. We call this approach the Laplace Approximation-based Graph Structure (LAGS) method, and show empirically that it improves the recent state-of-the-art GNNs.

1 Introduction

Graph neural networks (GNNs) (Gori et al., 2005; Scarselli et al., 2008; Kipf & Welling, 2017; Hamilton et al., 2017; Velickovic et al., 2018) rely on graph structure provided with the data, but such structures may not always truly represent the underlying dependencies. Often they are simply noisy and unreliable approximations of the true graph structure. Even when accurate, their topology may conflict with the chosen GNN architecture. For example, heterophilic graphs, where dissimilar nodes are connected, and graphs with long-range dependencies have been shown to perform poorly under most GNN models (Zhu et al., 2020; Alon & Yahav, 2021; Topping et al., 2022; NT & Maehara, 2019; Oono & Suzuki, 2019; Li et al., 2018). A common remedy is to change the architecture to better align with the data (Li et al., 2019; Chen et al., 2020a; Xu et al., 2018; Liu et al., 2020; Pei et al., 2020; Zhu et al., 2020), yet this still cannot resolve noisy and incorrect graph structures.

These problems can be addressed by learning the graph structure, but this is challenging because it involves searching over a large discrete space of all possible graphs. To reduce the cost, prior works often introduce explicit assumptions about the graph, enforced through rigid constraints. For example, some works induce global sparsity by penalizing the adjacency matrix (Luo et al., 2021; Jin et al., 2020), restricting edges to similar nodes (Fatemi et al., 2021; Liu et al., 2022; Gu et al., 2023), or limiting the graph to a predefined random-graph family (Zhang et al., 2019; Ma et al., 2019; Lu et al., 2025; Yang et al., 2024). However, determining the appropriate constraints for a given dataset, task, and GNN architecture is non-trivial and can contradict with the unknown optimal graph. Ideally, constraints should emerge automatically through regularization, guiding the search without relying on predefined assumptions about the structure. In this paper, we present such a method.

We propose using the marginal likelihood of the GNN as a simple solution to learn the graph structure. Marginal likelihood has long been used in Gaussian graphical models (Giudici & Green, 1999; Carvalho et al., 2007; Dobra et al., 2004), where the log-determinant of the covariance penalizes overly dense graphs (Dempster, 1972). Our main contribution is to extend this principle to GNNs via a variational formulation, where Laplace's method produces a marginal likelihood objective that automatically regularizes graph learning. Maximizing this marginal likelihood produces improved graph structures and enhances model performance (Fig. 1). Although defined over discrete graphs, the objective can be efficiently optimized using the Gum-



(a) GNN trained with a noisy graph leads to poor predictions.

(b) Better graph can be learned by maximizing marginal likelihood.

Figure 1: GNN performance is sensitive to graph quality. (a) When trained on graph $\mathbf{A}$, noisy connections (gray) can cause incorrect predictions (circled in red). (b) Learning the graph structure by maximizing the marginal likelihood removes noisy edges *(left)* and yields a refined topology *(right)*, improving accuracy.

bel–Softmax trick. We refer to this approach as the Laplace Approximation-based Graph Structure (LAGS) learning.

Our contributions are as follows: (1) we introduce LAGS, a simple, GNN-agnostic framework that leverages marginal likelihood for graph structure learning (Sec. 3); (2) we show significant performance gains from LAGS, with up to 8% higher accuracy on heterophilic graphs and up to 1% over the recent state-of-the-art GCN (Luo et al., 2024), which already outperforms graph transformers (Sec. 4.1, 4.2); and (3) we demonstrate through ablations that marginal likelihood effectively regularizes graph learning and learns edge probabilities aligned with edge importance (Sec. 4.3, 4.4).

2 Graph Structure Learning for GNNs

Consider the node classification task on a graph with N nodes. Each node i is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$, and for nodes in the training set we additionally observe a one-hot label vector $\mathbf{y}_i \in \{0,1\}^C$, where C is the number of classes. The observed (possibly noisy) graph structure is represented as a binary adjacency matrix $\mathbf{A}_0$ of size $N \times N$, where (i,j)'th entry of 1 denotes an edge between node i and j, while 0 indicates its absence. We denote the nodes' data as $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$, where the node features $\mathbf{X} \in \mathbb{R}^{N \times d}$ has $\mathbf{x}_i^\top$ as rows and node labels $\mathbf{Y} \in \{0,1\}^{N \times C}$ has $\mathbf{y}_i^\top$ as rows. Our objective is to jointly learn a refined graph $\mathbf{A}$ and GNN parameters $\boldsymbol{\theta}$ for predicting unlabeled node classes.

2.1 Graph Neural Networks

Graph neural networks (GNNs) are models that learn node representations by aggregating and transforming the node features using the input graph structure. A graph level representation can also be obtained by further aggregating over the learned node representations. Different GNN architectures are distinguished by their choice of aggregation and transformation functions. Here we consider two GNN variants: Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) and GraphSAGE (Hamilton et al., 2017), which continue to be powerful and widely used models for learning on graph (Luo et al., 2024; Wang et al., 2024; Dinh et al., 2025). Let us denote the learned node representations from layer l as $\mathbf{Z}^{(l)} \in \mathbb{R}^{N \times d_l}$, where d_l is the width of the layer.

Graph Convolutional Networks (GCNs) combine aggregation and transformation in a single step: $\mathbf{Z}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{Z}^{(l-1)}\boldsymbol{\Theta}^{(l)})$, where $\hat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}$, $\mathbf{D}$ is the degree matrix of $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$, $\boldsymbol{\Theta}^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ are the layer weights, $\sigma(\cdot)$ is a nonlinearity (e.g. ReLU), and $\mathbf{Z}^{(0)} = \mathbf{X}$. In contrast, GraphSAGE applies separate transformations to a node's own features and its neighbors. After aggregating neighbor features (e.g. by summation), the result is concatenated with the node's features before transformation: $\mathbf{Z}^{(l)} = \sigma(\boldsymbol{\Theta}^{(l)} \cdot \mathrm{Concat}(\mathbf{A}\mathbf{Z}^{(l-1)}, \mathbf{Z}^{(l-1)}))$, with $\boldsymbol{\Theta}^{(l)} \in \mathbb{R}^{2d_{l-1} \times d_l}$ as the layer weights.

For node classification, predictions $\hat{\mathbf{Y}}$ are obtained by applying the softmax function in the final layer. The GNN parameters, denoted $\boldsymbol{\theta} = \{\boldsymbol{\Theta}^{(l)}\}_{l=1}^{L}$, are learned by minimizing the regularized empirical loss, whose dependence on the graph structure $\mathbf{A}$ we make explicit:

$$\bar{\ell}(\boldsymbol{\theta}; \mathcal{D}, \mathbf{A}) = \ell(\boldsymbol{\theta}; \mathcal{D}, \mathbf{A}) + r(\boldsymbol{\theta}). \tag{1}$$

For classification, $\ell(\boldsymbol{\theta}; \mathcal{D}, \mathbf{A})$ is typically the average cross-entropy loss between the true label and the prediction over the whole data. The regularizer $r(\boldsymbol{\theta})$ may be explicit or implicit. We distinguish $\mathbf{A}$, the graph structure used in the GNN, from $\mathbf{A}_0$, the observed graph structure, to highlight that they need not be the same. In standard GNNs, the graph structure $\mathbf{A}$ is simply taken to be the observed graph $\mathbf{A}_0$.

2.2 Graph Structure Learning for GNNs

Standard use of GNNs assumes that the observed graph structure $\mathbf{A}_0$ encodes meaningful relationships among the nodes and fixes the computation graph $\mathbf{A}$ to $\mathbf{A}_0$. However, this assumption is often problematic: $\mathbf{A}_0$ may be noisy or inaccurate, and even when it is not, it may still be misaligned with the chosen GNN architecture. Heterophilic graphs and graphs with long-range interactions have, for instance, been shown to perform poorly with many GNNs (Zhu et al., 2020; Alon & Yahav, 2021; Topping et al., 2022; NT & Maehara, 2019; Oono & Suzuki, 2019; Li et al., 2018). Learning graph structure provides a remedy by preserving and augmenting only the most relevant relationships while avoiding the introduction of additional noise. Throughout, we denote the unknown graph structure by $\mathbf{A}$. The task of jointly learning $\mathbf{A}$ and the GNN parameters $\boldsymbol{\theta}$ is known as the graph structure learning (GSL) problem. We review some existing approaches to the problem and their challenges.

Existing solutions to the GSL problem can be broadly categorized into attention-based and direct approaches. Attention-based methods learn node repesentations and add or remove edges according to a chosen similarity metric (Velickovic et al., 2018; 2020; Chen et al., 2020b; Huang et al., 2020; Wu et al., 2022; Gu et al., 2023; In et al., 2024; Zhao et al., 2023), whereas direct methods treat the graph structure itself as learnable parameters (Zhang et al., 2019; Franceschi et al., 2019; Hasanzadeh et al., 2020; Jin et al., 2020; Chen et al., 2020b; Wang et al., 2021; Zhao et al., 2023). Although parameter-efficient, attention-based methods construct edges by linking nodes with similar representations under a fixed metric. This requires the node representations to align with the graph structure and makes it difficult to incorporate structural priors (e.g. from the observed graph) as parameters are not defined directly on the adjacency. Our method follows the direct learning approach.

Constraining the graph search space. To address the combinatorial space of possible graphs, many GSL methods restrict the learning process through assumptions about the target graph, typically implemented via additional loss terms or structural constraints. Common strategies include penalizing edges between distant nodes to promote smoothness (Yang et al., 2019; Chen et al., 2020b; Jin et al., 2020), minimizing the l_0 or l_1 norm of the adjacency matrix to encourage sparsity (Luo et al., 2021; Jin et al., 2020), or imposing low-rank constraints to promote homophily (Luo et al., 2021). In attention-based methods, sparsity is often enforced by restricting edges to the k-nearest neighbors or ϵ-distance neighbors according to similarity or attention scores (Chen et al., 2020b; Fatemi et al., 2021; Liu et al., 2022; In et al., 2024). Another common strategy for reducing the search space is to define a distribution over graphs and impose a prior, as in many variational GSL methods (Zhang et al., 2019; Lu et al., 2025; Yang et al., 2024). However, selecting constraints that are appropriate for a given dataset, task, and GNN architecture is challenging, and such choices may conflict with the (unknown) optimal graph structure.

Variational approaches to graph learning. Variational inference is widely used to capture uncertainty in unknown parameters by approximating their posterior with a tractable distribution (Blei et al., 2017). In GSL, this idea is applied to infer a distribution over graph structures (or edges) rather than a single deterministic graph. In this setting, regularization is induced through the choice of variational family and graph prior, which are often defined relative to the observed (potentially noisy) graph. A common choice is to parameterize the variational distribution using a random-graph model such as the stochastic block model (Zhang et al., 2019; Lu et al., 2025; Yang et al., 2024); however, this still requires selecting the appropriate graph family, with performance being heavily dependent on this choice. More expressive and flexible

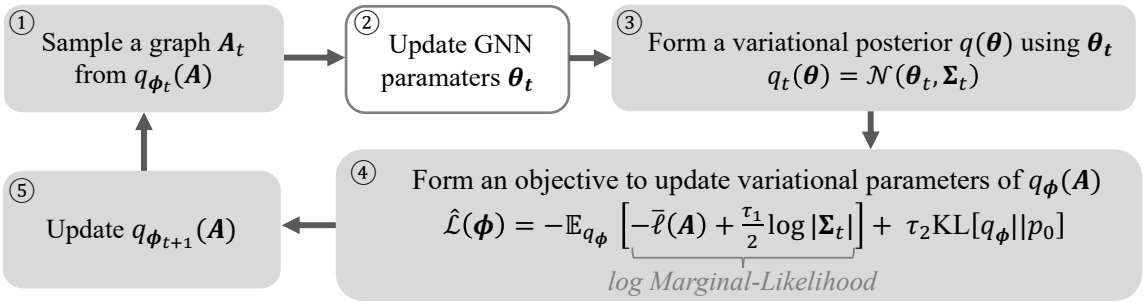


Figure 2: Overview of the Laplace Approximation-based Graph Structure (LAGS) learning framework. LAGS interleaves graph structure learning with standard GNN training: a graph is sampled, GNN parameters are updated, and a Laplace approximation forms a Gaussian posterior over parameters. The resulting marginal likelihood defines an objective that automatically regularizes graph learning, favoring graph structures that fit the data well and reduce model complexity.

variational approximations have been introduced, including node-copying (Pal et al., 2019; Komanduri & Zhan, 2021), pairwise-distance (Pal et al., 2020), and masking-based formulations (Hasanzadeh et al., 2020; Lu et al., 2025). Some approaches even directly learn per-edge Bernoulli parameters (Pantelis et al., 2020; Franceschi et al., 2019; Zhao et al., 2023), but this increased flexibility also enlarges the effective search space, still requiring additional regularization to keep the problem tractable.

A key challenge in GSL is the lack of a universal graph regularizer that generalizes across datasets, tasks, and GNN architectures. We address this by proposing a regularizer derived from the marginal likelihood of the GNN, rendering it agnostic to these factors. While prior variational GSL methods often neglect the posterior over GNN parameters or only capture it indirectly through techniques with a Bayesian interpretation (e.g. Monte Carlo dropout (Gal & Ghahramani, 2016)), we argue that this posterior contains valuable information and can be explicitly exploited via the marginal likelihood to automatically regularize graph learning.

3 Laplace Approximation-based Graph Structure (LAGS)

We introduce the *Laplace Approximation-based Graph Structure (LAGS)* learning method, a GNN-agnostic framework that jointly learns GNN parameters and graph structure through marginal likelihood maximization, automatically regularizing graph learning without the need for heuristic-based constraints. A variational formulation with Laplace's method yields the objective, which is optimized over discrete graphs using the Gumbel-Softmax trick. An overview of the method is shown in Fig. 2, with pseudo-code provided in Alg. 1. Before introducing the method in Secs. 3.2 to 3.4, we first discuss the motivation for using marginal likelihood in Sec. 3.1.

3.1 Motivation for marginal likelihood

The marginal likelihood (or evidence) is the integral of the likelihood with respect to the prior over parameters. It has been used in Gaussian graphical models (Giudici & Green, 1999; Carvalho et al., 2007; Dobra et al., 2004) to search over graph structures, where the precision matrix (inverse covariance) encodes conditional dependencies, effectively reducing graph learning to the classical covariance selection problem (Dempster, 1972). The objective is a marginalization over the observed Gaussian-distributed samples, and its dual corresponds to maximizing the entropy, $\log|\boldsymbol{\Sigma}|$, under constraints that enforce agreement with the empirical covariance for known entries (Good, 1963; Dempster, 1972; Uhler, 2017). Maximizing the entropy naturally induces sparsity in the graph: a larger determinant of the covariance implies a sparser precision matrix and thus a sparser graph. We extend this principle to GNNs, where the covariance is instead defined over the parameters, but the same effect holds–a larger determinant reflects a simpler model with weakly coupled parameters. Graphs that simplify the data allow the model to remain simple, while dense or heterogeneous graphs entangle features, increasing parameter coupling and reducing the determinant.

Marginal likelihood is also central to Bayesian model selection for model comparison (MacKay, 1992; Jeffreys, 1939; Gull, 1988). It provides a principled trade-off between data fit and model complexity, also known as Occam's razor. Consider a model $\mathcal{M}$ with parameters $\boldsymbol{\theta}$ and a Gaussian approximation to the posterior, as is common in variational inference (Blei et al., 2017). Under this approximation, Laplace's method gives the following approximation for the log marginal-likelihood: $\log p(\mathcal{D}|\mathcal{M}) \propto \log p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) + \frac{1}{2}\log|\boldsymbol{\Sigma}|$, where $\boldsymbol{\Sigma}$ is the posterior covariance of the parameters (MacKay, 1992). The first term measures data fit, whereas the log-determinant term captures the posterior volume (entropy) and penalizes model complexity. Previous work has shown that maximizing the marginal likelihood can be used to learn hyperparameters efficiently in an online manner (Immer et al., 2021). We show that it is also effective for jointly learning graph structure and GNN parameters: the marginal likelihood naturally guides the search toward informative graph structures that maintain strong fit while controlling model complexity.

3.2 Variational graph structure learning

We formulate learning of both the binary adjacency matrix $\mathbf{A} \in \{0,1\}^{N\times N}$ and GNN parameters $\boldsymbol{\theta}$ as a variational inference task. The joint posterior is approximated with a factorized variational distribution as follows:

$$q(\boldsymbol{\theta}, \mathbf{A}) = q(\boldsymbol{\theta})q(\mathbf{A}). \tag{2}$$

In contrast to conventional (non-variational) GNNs where $\boldsymbol{\theta}$ is learned as a point-estimate, here $\boldsymbol{\theta}$ is treated as a random variable under $q(\boldsymbol{\theta})$.

We assume that the posterior over $\boldsymbol{\theta}$ is a Gaussian with mean $\mathbf{m}$ and covariance $\boldsymbol{\Sigma}$:

$$q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, \boldsymbol{\Sigma}), \tag{3}$$

where we denote the parameter by $\boldsymbol{\lambda} = (\mathbf{m}, \boldsymbol{\Sigma})$. For the graph structure posterior $q(\mathbf{A})$, we assume it is a mean-field Bernoulli distribution over all N^2 potential edges, with N being the number of nodes:

$$q_{\boldsymbol{\phi}}(\mathbf{A}) = \prod_{i=1}^{N}\prod_{j=1}^{N} \mathrm{Ber}(a_{ij}|\pi_{ij} = \sigma(\phi_{ij})), \tag{4}$$

where $a_{ij} \in \{0,1\}$ is the (i,j)'th entry of $\mathbf{A}$ and $\pi_{ij} \in (0,1)$ is the probability that $a_{ij} = 1$ (an edge exists between node i and j). To parameterize the edge probabilities π_{ij} in an unconstrained way during optimization, we introduce real-valued logits $\phi_{ij} \in \mathbb{R}$ and set $\pi_{ij} = \sigma(\phi_{ij})$, where $\sigma(\cdot)$ is the sigmoid function. We denote $\boldsymbol{\phi}$ as the vector of logits that parameterizes $q_{\boldsymbol{\phi}}(\mathbf{A})$.

3.3 Joint learning of $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$

To learn the posterior parameters, we will minimize the following variational objective with temperatures $\tau_1, \tau_2 > 0$:

$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\lambda}},\, \boldsymbol{\phi}(\boldsymbol{\theta},\mathbf{A})}\big[\ell(\boldsymbol{\theta}, \mathbf{A}; \mathcal{D})\big] + \tau_1 \mathrm{KL}\big[q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})||p_0(\boldsymbol{\theta})\big] + \tau_2 \mathrm{KL}\big[q_{\boldsymbol{\phi}}(\mathbf{A})||p_0(\mathbf{A})\big], \tag{5}$$

where $\ell(\boldsymbol{\theta}, \mathbf{A}; \mathcal{D})$ is the unregularized empirical loss (from Eq. 1), and $p_0(\boldsymbol{\theta})$ and $p_0(\mathbf{A})$ are the prior distributions for $\boldsymbol{\theta}$ and $\mathbf{A}$ respectively. We propose to perform a double-loop minimization of Eq. 5 where, at each iteration t given $\boldsymbol{\phi}_t$, we first minimize $\boldsymbol{\lambda}$ using a cheap Laplace approximation to get $\boldsymbol{\lambda}_t(\boldsymbol{\phi}_t)$ in the inner loop, and then use this solution to find the next $\boldsymbol{\phi}_{t+1}$. This is shown below,

$$\boldsymbol{\lambda}_t(\boldsymbol{\phi}_t) \leftarrow \arg\min_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}_t), \qquad \boldsymbol{\phi}_{t+1} \leftarrow \arg\min_{\boldsymbol{\phi}} \widehat{\mathcal{L}}(\boldsymbol{\lambda}(\boldsymbol{\phi}_t), \boldsymbol{\phi}), \tag{6}$$

where we denote the surrogate $\widehat{\mathcal{L}}$ used in the outer loop. The surrogate will utilize the marginal likelihood approximation, which is the main contribution of this work.

We start by discussing the minimization with respect to $\boldsymbol{\lambda} = (\mathbf{m}, \boldsymbol{\Sigma})$ given $\boldsymbol{\phi}_t$. We rely on the result by Khan & Rue (2023, App. C.1) who derive Laplace's method as a special case of variational learning,

which we will use to simplify the training. Essentially, they show that approximating the expectation with respect to the Gaussian $q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$ at its mean $\mathbf{m}$, via the so-called delta method, reduces variational learning to Laplace approximation. We provide detailed derivative in Appendix B and summarize them here. First, we approximate the expectation with respect to $q_{\boldsymbol{\phi}_t}(\mathbf{A})$ at one sample, denoted as $\mathbf{A}_t \sim q_{\boldsymbol{\phi}_t}(\mathbf{A})$, and expand the KL term, to obtain the following objective,

$$\begin{aligned}\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}_t) &\approx \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\left[\ell(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})\right] + \tau_1 \mathrm{KL}\big[q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) || p_0(\boldsymbol{\theta})\big] + \text{const.} \\ &= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\left[\bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})\right] - \tau_1 \tfrac{1}{2} \log |\boldsymbol{\Sigma}| + \text{const.}\end{aligned} \tag{7}$$

This rewrites the problem in terms of the regularized empirical loss $\bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})$ from Eq. 1 where the regularizer is defined as $r(\boldsymbol{\theta}) = -\tau_1 \log p_0(\boldsymbol{\theta})$. The objective can be further simplified to the Laplace approximation by using the second-order Taylor approximation at $\mathbf{m}$, giving us:

$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}_t) \approx \bar{\ell}(\mathbf{m}, \mathbf{A}_t; \mathcal{D}) - \tau_1 \tfrac{1}{2} \log |\boldsymbol{\Sigma}| + \mathrm{Tr}\left[\mathbf{H}_{\mathbf{m}} \boldsymbol{\Sigma}\right],$$

where $\mathbf{H}_{\mathbf{m}} = \nabla^2 \bar{\ell}(\mathbf{m}, \mathbf{A}_t; \mathcal{D})$ is the Hessian at $\mathbf{m}$. Optimization with respect to $\mathbf{m}$ reduces to minimizing $\bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})$, whose minimizer is $\boldsymbol{\theta}_t$. We therefore set $\mathbf{m}_t = \boldsymbol{\theta}_t$. The optimization with respect to $\boldsymbol{\Sigma}$ is available in closed form by setting $\boldsymbol{\Sigma}_t = \nabla^2 \bar{\ell}(\boldsymbol{\theta}_t, \mathbf{A}_t; \mathcal{D})^{-1}$, equivalent to $\mathbf{H}_{\boldsymbol{\theta}_t}^{-1}$; details on Hessian approximations are given in Appendix B. By using the delta method, we have reduced the optimization over $\boldsymbol{\lambda}$ to the standard Laplace approximation.

We now propose the surrogate loss used in the outer loop to optimize with respect to $\boldsymbol{\phi}$ given $\boldsymbol{\lambda}_t(\boldsymbol{\phi}_t)$. Our main idea is to utilize the dependence of $\boldsymbol{\lambda}_t(\boldsymbol{\phi}_t)$ on the sample $\mathbf{A}_t \sim q_{\boldsymbol{\phi}_t}(\mathbf{A})$, and replacing $\mathbf{A}_t$ with a free variable $\mathbf{A}$. We show this by first writing the objective (Eq. 5) at the solution $\boldsymbol{\lambda}_t(\boldsymbol{\phi}_t)$ and $\boldsymbol{\phi}_t$, substituting $\mathbf{m}_t = \boldsymbol{\theta}_t$ and $\boldsymbol{\Sigma}_t = \nabla^2 \bar{\ell}(\boldsymbol{\theta}_t, \mathbf{A}_t; \mathcal{D})^{-1}$ to make the dependence on $\mathbf{A}_t$ explicit. In our proposed surrogate loss, $\mathbf{A}_t$ is replaced by a free variable (shown in red):

Solution of Eq. 5 at t:

$$\mathcal{L}(\boldsymbol{\lambda}_t(\boldsymbol{\phi}_t), \boldsymbol{\phi}_t) \approx \bar{\ell}(\boldsymbol{\theta}_t, \mathbf{A}_t; \mathcal{D}) + \tau_1 \tfrac{1}{2} \log |\nabla^2 \bar{\ell}(\boldsymbol{\theta}_t, \mathbf{A}_t; \mathcal{D})| + \tau_2 \mathrm{KL}\big[q_{\boldsymbol{\phi}_t}(\mathbf{A}) || p_0(\mathbf{A})\big]$$

Proposed surrogate loss:

$$\widehat{\mathcal{L}}(\boldsymbol{\lambda}_t(\boldsymbol{\phi}_t), \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{A})}\left[\bar{\ell}(\boldsymbol{\theta}_t, \mathbf{A}; \mathcal{D}) + \tau_1 \tfrac{1}{2} \log |\nabla^2 \bar{\ell}(\boldsymbol{\theta}_t, \mathbf{A}; \mathcal{D})|\right] + \tau_2 \mathrm{KL}\big[q_{\boldsymbol{\phi}}(\mathbf{A}) || p_0(\mathbf{A})\big]. \tag{8}$$

The expectation term above is simply the negative log marginal-likelihood obtained by the Laplace approximation at $\boldsymbol{\theta}_t$ but the graph adjacency is now treated as a free variable $\mathbf{A}$ that can be optimized.

The advantage of this objective is that it incorporates the Hessian at $\boldsymbol{\theta}_t$ that automatically regularizes the graph learning process. The log-determinant of the Hessian is the negative of the log-determinant of the covariance, so minimizing Eq. 8 effectively maximizes the covariance determinant. As discussed in Sec. 2.2, this determinant reflects the posterior's entropy and provides a measure of model complexity (Dempster, 1972; Good, 1963). A larger value indicates a simpler model with higher posterior entropy, where parameter distributions are flatter and more independent. This, in turn, favors graph structures that simplify the data and can be explained by simple models, while discouraging dense or heterophilic structures that entangle class representations and require more complex models. The necessity of including the Hessian is empirically validated in Sec. 4.3, where removing it reduces the objective to likelihood maximization and results in degraded performance.

3.4 Learning Discrete Graph Structures

Since $\mathbf{A}$ is dicrete, standard gradient-based optimization methods cannot be used to update $\boldsymbol{\phi}$. One approach is to use the score function (REINFORCE) estimator which transforms the gradient of the expectation into the expectation of the gradient using the log-derivative trick, but such approach is known to have high variance. We instead propose to use the Gumbel-Softmax trick, which uses the concrete distribution to relax discrete random variables. The relaxed variable of a_{ij} (corresponding to the edge between node i and j) is denoted as $a_{ij}^r \in (0, 1)$, with probability distribution given by the binary Concrete distribution (from

Algorithm 1 Laplace Approximation-based Graph Structure (LAGS)

Input: training data $\mathcal{D}$, GNN learning rate η, graph learning rate γ, graph prior $p_0(\mathbf{A})$, temperatures (τ_1, τ_2), burn-in epoch B, update freq F, steps K
$\boldsymbol{\phi}_0 \leftarrow$ initialize such that $\mathrm{KL}[q_{\boldsymbol{\phi}_0}(\mathbf{A})||p_0(\mathbf{A})] = 0$
$\boldsymbol{\theta}_0 \leftarrow$ initialize GNN parameters
for each epoch **do**
  $\boldsymbol{\theta}_t \leftarrow$ optimize for $\bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})$ (Eq. 1) with $\mathbf{A}_t \sim q_{\boldsymbol{\phi}_t}(\mathbf{A})$
  for $k = 1, ..., K$ **every** F epochs after burn-in B **do**
    $\widehat{\mathcal{L}}(\boldsymbol{\lambda}(\boldsymbol{\phi}_t), \boldsymbol{\phi}) \leftarrow$ Laplace approx. marginal likelihood surrogate loss (Eq. 8)
    $\mathbf{g}_{\boldsymbol{\phi}} \leftarrow$ gradient of $\widehat{\mathcal{L}}(\boldsymbol{\lambda}(\boldsymbol{\phi}_t), \boldsymbol{\phi})$ w.r.t. $\boldsymbol{\phi}$ calculated with Gumbel-Softmax trick
    $\boldsymbol{\phi}_{t+1} \leftarrow$ optimizer step using normalized $\mathbf{g}_{\boldsymbol{\phi}}$
  end for
end for
Return $q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}), q_{\boldsymbol{\phi}}(\mathbf{A})$

Maddison et al. (2017, App. B)):

$$a_{ij}^r = \frac{1}{1 + \exp(-\frac{\log \alpha_{ij} + L}{\tau})}, \qquad p(a_{ij}^r(\pi_{ij})) = \frac{\tau \alpha (a_{ij}^r(\pi_{ij}))^{-\tau-1} (1 - (a_{ij}^r(\pi_{ij})))^{-\tau-1}}{(\alpha (a_{ij}^r(\pi_{ij}))^{-\tau} (1 - (a_{ij}^r(\pi_{ij})))^{-\tau})^2}. \tag{9}$$

where $\alpha_{ij} = \frac{\pi_{ij}}{1-\pi_{ij}}$ and $L = \log \frac{\epsilon_{ij}}{1-\epsilon_{ij}}$ is a sample from the Logistic distribution, with $\epsilon_{ij} \sim \mathcal{U}(0, 1)$. The temperature τ controls the degree of relaxation. To make the relaxed a_{ij}^r fully discrete, we use straight-through estimator (Bengio et al., 2013) which thresholds a_{ij}^r to make it binary in the forward pass, while the gradient is calculated using the continuous a_{ij}^r. In the forward pass, the graph $\mathbf{A}$ is sampled from $q_{\boldsymbol{\phi}}(\mathbf{A})$ by sampling each edge a_{ij}.

4 Experiments

We evaluate LAGS on node classification with two widely used GNN architectures: Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) and GraphSAGE (Hamilton et al., 2017), which remain standard models for graph learning (Luo et al., 2024; Wang et al., 2024; Dinh et al., 2025). Our experiments examine: (1) the performance gains LAGS brings to base GNNs (Sec. 4.1); (2) the improvement of LAGS on state-of-the-art GNNs (Sec. 4.2); (3) the advantage of marginal likelihood in regularizing graph learning (Sec. 4.3); and (4) LAGS's ability to learn edge distribution that distinguish informative from noisy edges (Sec. 4.4). The implementation is provided in the supplementary material and will be released publicly upon acceptance. Experimental details and hyperparameters are given in Appendix D.

Datasets. We evaluate on standard benchmarks spanning homophilic and heterophilic networks. Citation graphs (Cora, Citeseer, Pubmed) use splits from Yang et al. (2016); Kipf & Welling (2017) (Table 1) and Luo et al. (2024) (Table 2). Social networks (BlogCatalog, Flickr) follow Huang et al. (2017); Zhao et al. (2021), and Wikipedia graphs (Squirrel, Chameleon, Roman-empire) follow Platonov et al. (2023). For further details, see Appendix D.1.

4.1 Performance Gains from LAGS

LAGS improves GNN performance. We assess the performance improvements achieved by applying LAGS to GCN and GraphSAGE, with results summarized in Table 1. For both models, LAGS consistently improve upon the base model, with the largest gains on heterophilic graphs (BlogCatalog, Flickr, Roman-Empire), where LAGS-GCN improves accuracy by about 8%. Even on homophilic benchmarks (Cora, Citeseer, Pubmed), LAGS still produce measurable gains. Statistical significance is confirmed by p-values in Appendix D.5, and an ablation on graph priors is given in Appendix E.1. Notably, GCN benefits more from LAGS than GraphSAGE, likely because GCN is more sensitive to graph structure: its one-step convolution

Table 1: Node classification results (%) using standard hyperparameters. "OOM" and "OOT" denote out of memory and out of time (within 12 hours), respectively.

Dataset	Cora	Citeseer	Pubmed	BlogCatalog	Flickr	Roman Empire
GCN	81.31±0.46	71.19±0.51	78.62±0.42	75.48±0.37	63.71±0.37	52.52±0.38
+LAGS	↑ 1% **82.63**±0.24	↑ 1% **72.44**±1.54	↑ .3% **78.88**±0.29	↑ 9% **84.93**±0.58	↑ 10% **73.91**±0.30	↑ 8% **60.72**±0.42
GraphSAGE	80.52±0.43	70.57±0.83	78.31±0.31	91.45±0.28	78.06±0.90	77.16±0.68
+LAGS	↑ 1% **81.10**±0.57	↑ .7% **71.31**±0.69	↑ .3% **78.64**±0.25	↑ .2% **91.66**±0.30	↑ 1% **79.19**±0.62	↑ 1% **78.46**±0.58
LDS	84.24±0.40	74.79±0.42	OOM	OOT	OOM	OOM
IDGL	83.41±1.22	72.26±1.09	OOM	88.63±0.08	86.03±0.26	OOM
IDGL-Anch	81.14±0.67	70.62±0.99	81.56±1.25	85.00±0.59	69.83±1.11	33.15±0.97
NodeFormer	79.59±0.83	70.78±1.61	78.99±1.53	43.91±22.07	17.11±0.67	49.19±4.95
WSGNN	83.62±0.53	73.04±0.74	OOM	91.86±0.56	80.76±2.51	OOM
SUBLIME	77.90±1.14	73.10±0.54	80.76±0.69	88.90±0.87	86.88±0.46	60.95±0.30
+unGSL	79.20±1.38	71.13±1.37	78.10±0.92	92.14±0.85	86.80±0.20	62.45±0.38
GraphGLOW	82.12±0.85	73.44±0.69	79.56 ±0.42	26.04 ±1.57	18.14±0.47	39.19±0.71

directly mixes node features with their neighbors, making it more vulnerable to suboptimal graphs (Zhu et al., 2020) and thus more responsive to structure learning.

LAGS is competitive with other GSL methods. We evaluate how LAGS position itself with other state-of-the-art graph structure learning algorithms (Zhou et al., 2023). Results in Table 1 show that LAGS's GCN and GraphSAGE variants are competitive with these methods, despite its simplicity and generality. While LDS (Franceschi et al., 2019), IDGL (Chen et al., 2020b) and WSGNN (Lao et al., 2022) achieve higher accuracy, they face scalability issues on larger datasets. IDGL-Anch, although more scalable, underperforms both variants of LAGS. Similarly, NodeFormer (Wu et al., 2022), despite being a transformer-based model, is outperformed by LAGS. SUBLIME (Liu et al., 2022) and unGSL (Han et al., 2025) are competitive but do not consistently surpass LAGS, whereas GraphGLOW (Zhao et al., 2023) struggles on heterophilic graphs. Moreover, compared to LAGS, these methods rely on more complex architectures: NodeFormer requires a transformer backbone, WSGNN trains two GNNs, SUBLIME adds an auxiliary learner, unGSL depends on another GSL method (e.g. SUBLIME), GraphGLOW relies on transfer learning from other graph data, and LDS requires validation data for graph learning. See Appendix D.7 for additional details.

LAGS can be scalable. We show that LAGS can be scaled by batching and applying a sparse prior on the graph. Directly learning parameters for all N^2 edges can be intractable, so we restrict learning to a candidate set formed by the union of edges from a k-nearest neighbor (kNN) graph and the observed graph. We evaluate this setting on ogbn-arxiv (Hu et al., 2020) (169k nodes, 1.16M edges) using GraphSAGE, which naturally supports batching. Using a kNN graph ($k = 5$) combined with the observed graph, LAGS improves GraphSAGE performance from 62.51% (±0.21) to 63.46% (±0.06), and training with a batch size of 1024 nodes required ∼4 hours and 38 GB of GPU memory, but both can be further optimized with sparser priors and different batch sizes. Additional details are provided in Appendix E.2.

4.2 Improving upon State-of-the-Art GNNs with LAGS

We evaluate the improvement LAGS can bring to state-of-the-art GNN. Recent work by Luo et al. (2024) shows that expanding the set of hyperparameters—such as batch normalization, layer normalization, and residual connections—can significantly improve the performance of classic GNNs (e.g. GCN), allowing them to surpass many state-of-the-art graph transformers. Following their setup, we adopt these enhancements for GCN (denoted as GCN*) and demonstrate that LAGS can further improve its performance. As shown in Table 2, LAGS provide additional boost to GCN*, and we include additional analysis of the learned graph in Appendix D.8 and p-values in Appendix D.5.

Table 2: Node classification results (%) using SoTA hyperparameters and setup from Luo et al. (2024). "*" denotes difference in model / data split from Table 1.

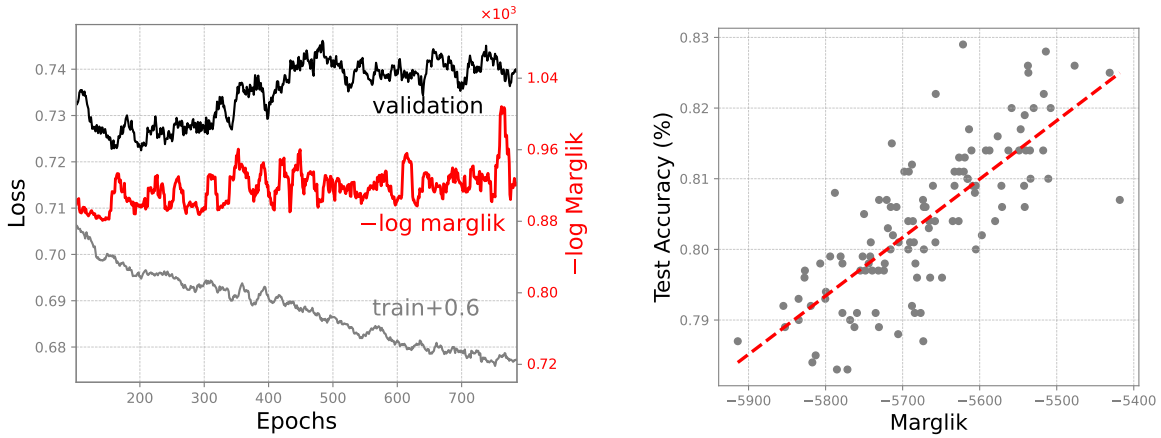
Dataset	Cora*	Citeseer*	Pubmed*	Squirrel	Chameleon
GCN*	84.21 ± 0.96	72.47 ± 0.44	80.10 ± 0.76	44.21 ± 1.59	44.35 ± 4.30
+LAGS	↑ .4% **84.61** ± 0.74	↑ .6% **73.10** ± 0.48	↑ .2% **80.33** ± 0.84	↑ .1% **44.64** ± 2.02	↑ 1% **45.32** ± 4.17

4.3 Ablation on marginal likelihood

To analyze the regularization effect of marginal likelihood in graph structure learning, we conduct an ablation comparing marginal likelihood (LAGS) with likelihood as the objective. The key distinction is the log-determinant of the Hessian term present in the marginal likelihood. As discussed in Sec. 3.3, this term quantifies the entropy of the GNN parameter posterior, which we hypothesize to be crucial for regulating the graph learning process. Removing this term reduces the surrogate loss (Eq. 8) to likelihood maximization. Results with GCN (Table 3) show that marginal likelihood yields consistently higher gains, while likelihood alone can even harm performance (as in Citeseer and Pubmed), confirming the regulating effect of the log-determinant term.

Table 3: Node classification accuracy (%) with GCN, comparing vanilla training, graph structure learning with likelihood objective, and with marginal likelihood (LAGS-GCN).

Dataset	Cora	Citeseer	Pubmed	BlogCatalog	Flickr	Roman Empire
Vanilla	81.31 ± 0.46	71.19 ± 0.51	78.62 ± 0.42	75.48 ± 0.37	63.71 ± 0.37	52.52 ± 0.38
Likelihood	↑ 81.98 ± 0.17	↓ 70.62 ± 1.59	↓ 78.06 ± 0.67	↑ 83.89 ± 0.11	↑ 72.38 ± 0.67	↑ 60.64 ± 0.46
Marglik	↑ **82.63** ± 0.24	↑ **72.44** ± 1.54	↑ **78.88** ± 0.29	↑ **84.93** ± 0.58	↑ **73.91** ± 0.30	↑ **60.72** ± 0.42



(a) Training of LAGS-GCN on Cora

(b) GCNs trained on the graph checkpoints

Figure 3: Marginal likelihood is a reliable proxy for GNN generalization and graph quality. (a) For LAGS-GCN on Cora, the log marginal likelihood matches the validation loss (MA window=15), providing a reliable measure of generalization. (b) Vanilla GCNs trained on checkpoint graphs show that better graph structures (with higher test accuracy) correlate with marginal likelihood.

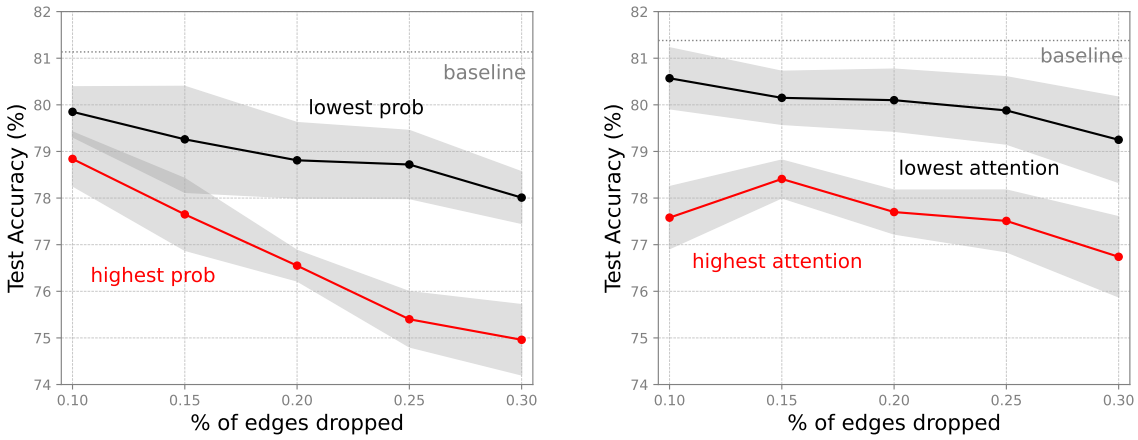
We further show that marginal likelihood serves as a reliable proxy for both model generalization and the quality of the learned graph structure. As illustrated in Fig.3(a), unlike the training loss (likelihood), which decreases monotonically, the marginal likelihood closely tracks the validation loss. To isolate the effect of the graph structure, we train vanilla GCNs using graph checkpoints obtained during LAG-GCN training. As shown in Fig.3(b), graphs with higher marginal likelihood consistently yield better test accuracy, indicating

that marginal likelihood is predictive of graph quality and downstream generalization.. Together, these results confirm that marginal likelihood provides a principled objective for graph structure learning that aligns with model generalizability.

4.4 Ablation on LAGS's learned edge probabilities

We examine whether the posterior edge probabilities learned by LAGS reflect edge importance. After training LAGS-GCN on Cora, we rank edges by their learned probabilities and construct perturbed graphs by removing varying fractions of either the highest- or lowest-probability edges. We then train vanilla GCNs on these perturbed graphs and evaluate how performance changes as high- versus low-probability edges are dropped. As shown in Fig. 4(a), removing high-probability edges causes a substantially faster degradation in accuracy than removing low-probability edges. This suggests that LAGS learns meaningful edge probabilities that correlate with edge importance.

We conduct the same edge-dropping experiment for Graph Attention Networks (GAT) (Velickovic et al., 2018), where edges are ranked by attention weights and the highest/lowest fractions are removed. Relative to GAT, removing high-probability edges under LAGS yields a steeper performance drop than removing high-attention edges under GAT. This indicates that LAGS produces edge probabilities that better reflect edge importance than GAT attention weights.



(a) Edge probabilities determined by LAGS-GCN

(b) Attention scores determined by GAT

Figure 4: LAGS learns edge probabilities that reflect informativeness. (a) Dropping high-probability edges from LAGS-GCN degrades performance more than low-probability ones. (b) GAT attention scores show weaker correlation, indicating LAGS captures edge importance more effectively.

5 Conclusion

We propose marginal likelihood as a simple and effective objective for graph structure learning in GNNs. Marginal likelihood naturally regularizes the structure learning process by favoring informative structures that simplify the data and can be explained by simple GNNs, while discouraging dense or heterophilic graphs that entangle node features and demand more complex models. Our framework, Laplace Approximation-based Graph Structure (LAGS), derives this objective via a variational formulation with Laplace's method and optimizes it over discrete graphs using the Gumbel-Softmax trick. Implemented with GCN and GraphSAGE, LAGS achieves up to 8% accuracy gains on heterophilic graphs and improves the state-of-the-art GCN* (Luo et al., 2024) by up to 1%. While computing Hessians may pose scalability challenges for large GNNs, efficient approximations offer promising directions, which we plan to explore in future work. We also aim to extend LAGS to attention-based graph learning methods as a potentially more scalable alternative.

References

Uri Alon and Eran Yahav. On the bottleneck of Graph Neural Networks and its practical implications. In *Proceedings of the International Conference on Learning Representations*, 2021.

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

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

Carlos M. Carvalho, Hélène Massam, and Mike West. Simulation of hyper-inverse wishart distributions in graphical models. *Biometrika*, 94(3), 2007. ISSN 00063444, 14643510.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep Graph Convolutional Networks. In *Proceedings of the International Conference on Machine Learning*, 2020a.

Yao Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for Graph Neural Networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33, 2020b.

Arthur P. Dempster. Covariance selection. *Biometrics*, 28(1):157–175, 1972.

Thi N. Dinh, Phu Pham, Giang L. Nguyen, Ngoc Thanh Nguyen, and Bay Vo. A hybrid citation recommendation model with sciBERT and GraphSAGE. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 55(2):852–863, 2025.

Adrian Dobra, Chris Hans, Beatrix Jones, Joseph R. Nevins, Guang Yao, and Mike West. Sparse graphical models for exploring gene expression data. *Journal of Multivariate Analysis*, 90(1):196–212, 2004.

Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for Graph Neural Networks. In *Proceedings of the International Conference on Machine Learning*, 2019.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2016.

Paolo Giudici and Peter J. Green. Decomposable graphical gaussian model determination. *Biometrika*, 86 (4), 1999.

Irving J. Good. Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables. *The Annals of Mathematical Statistics*, 34(3):911 – 934, 1963.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2005.

Roger B. Grosse and Ruslan Salakhutdinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2015.

Ming Gu, Gaoming Yang, Sheng Zhou, Ning Ma, Jiawei Chen, Qiaoyu Tan, Meihan Liu, and Jiajun Bu. Homophily-enhanced structure learning for graph clustering. In *Proceedings of the ACM International Conference on Information and Knowledge Management*. Association for Computing Machinery, 2023.

Stephen F. Gull. *Bayesian Inductive Inference and Maximum Entropy.* Springer Netherlands, 1988.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Shen Han, Zhiyao Zhou, Jiawei Chen, Zhezheng Hao, Sheng Zhou, Gang Wang, Yan Feng, Chun Chen, and Can Wang. Uncertainty-aware graph structure learning. In *Proceedings of the ACM on Web Conference*, New York, NY, USA, 2025. Association for Computing Machinery.

Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian Graph Neural Networks with adaptive connection sampling. In *Proceedings of the International Conference on Machine Learning*, 2020.

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 *Proceedings of the International Conference on Neural Information Processing Systems*, 2020.

Dianwen Huang, Pichao Chen, Runhao Zeng, Qiaoyong Du, Mingkui Tan, and Chuang Gan. Location-aware graph convolutional networks for video question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Xiao Huang, Jundong Li, and Xia Hu. Label informed attributed network embedding. In *Proceedings of the ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, 2017.

Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable marginal likelihood estimation for model selection in deep learning. In *Proceedings of the International Conference on Machine Learning*, 2021.

Alexander Immer, Tycho F.A. van der Ouderaa, Gunnar Ratsch, Vincent Fortuin, and Mark van der Wilk. Invariance learning in deep neural networks with differentiable laplace approximations. In *Advances in Neural Information Processing Systems*, 2022.

Yeonjun In, Kanghoon Yoon, Kibum Kim, Kijung Shin, and Chanyoung Park. Self-guided robust graph structure refinement. In *Proceedings of the ACM Web Conference*, 2024.

Harold Jeffreys. *The Theory of Probability.* Oxford Classic Texts in the Physical Sciences. 1939.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust Graph Neural Networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

Mohammad Emtiyaz Khan and Håvard Rue. The Bayesian learning rule. *Journal of Machine Learning Research*, 24, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations (ICLR)*, 2015.

Thomas Kipf and Max Welling. Semi-supervised classification with Graph Convolutional Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Aneesh Komanduri and Justin Zhijun Zhan. Neighborhood random walk graph sampling for regularized bayesian graph convolutional neural networks. *IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2021.

Danning Lao, Xinyu Yang, Qitian Wu, and Junchi Yan. Variational inference for training graph neural networks in low-data regime through joint structure-label estimation. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can GCNs go as deep as CNNs? In *The International Conference on Computer Vision*. IEEE, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper Graph Neural Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2020.

Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM Web Conference*, 2022.

Mingjie Lu, Zhaowei Liu, Pengda Wang, Haiyang Wang, and Dong Yang. A bayesian graph structure inference neural network based on adaptive connection sampling. *Applied Soft Computing*, 2025.

Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. Learning to drop: Robust graph neural network via topological denoising. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, 2021.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic GNNs are strong baselines: Reassessing GNNs for node classification. In *The Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. A flexible generative framework for graph-based semi-supervised learning. In *Advances in Neural Information Processing Systems*, 2019.

David J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proceedings of the International Conference on Learning Representations*, 2017.

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2010.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2015.

Hoang NT and Takanori Maehara. Revisiting Graph Neural Networks: All we have is low-pass filters. *ArXiv*, 2019.

Kenta Oono and Taiji Suzuki. Graph Neural Networks exponentially lose expressive power for node classification. *ArXiv*, 2019.

Kazuki Osawa, Satoki Ishikawa, Rio Yokota, Shigang Li, and Torsten Hoefler. Asdl: A unified interface for gradient preconditioning in pytorch, 2023.

Soumyasundar Pal, Florence Regol, and Mark J. Coates. Bayesian graph convolutional neural networks using node copying. *ArXiv*, abs/1911.04965, 2019.

Soumyasundar Pal, Saber Malekmohammadi, Florence Regol, Yingxue Zhang, Yishi Xu, and Mark Coates. Non parametric graph learning for bayesian graph neural networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

Elinas Pantelis, Edwin V. Bonilla, and Louis C. Tiao. Variational inference for Graph Convolutional Networks in the absence of graph data and adversarial settings. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations*, 2014.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*, 2020.

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at evaluation of gnns under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*, 2023.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

Caroline Uhler. Gaussian graphical models: An algebraic and geometric perspective. *arXiv: Statistics Theory*, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lió, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

Petar Velickovic, Lars Buesing, Marwin Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 2232–2244. Curran Associates, Inc., 2020.

Duo Wang, Yuan Zuo, Fengzhi Li, and Junjie Wu. LLMs as zero-shot graph learners: Alignment of GNN representations with LLM token embeddings. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. Graph structure estimation neural networks. In *Proceedings of the Web Conference 2021*, 2021.

Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, 2022.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the International Conference on Machine Learning*, 2018.

Dong Yang, Zhaowei Liu, Yingjie Wang, Jindong Xu, Weiqing Yan, and Ranran Li. Adaptive multi-channel bayesian graph neural network. *Neurocomputing*, 575:127260, 2024.

Liang Yang, Zesheng Kang, Xiaochun Cao, Di Jin, Bo Yang, and Yuanfang Guo. Topology optimization based Graph Convolutional Network. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2019.

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

Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Üstebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Wentao Zhao, Qitian Wu, Chenxiao Yang, and Junchi Yan. Graphglow: Universal and generalizable structure learning for graph neural networks. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

Zhiyao Zhou, Sheng Zhou, Bochao Mao, Xuanyi Zhou, Jiawei Chen, Qiaoyu Tan, Daochen Zha, Yan Feng, Chun Chen, and Can Wang. OpenGSL: A comprehensive benchmark for graph structure learning. In *Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in Graph Neural Networks: current limitations and effective designs. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020.

A Appendix

B Derivation of the Laplace approximation

We provide a detailed derivation of the Laplace approximation for the optimization of $\boldsymbol{\lambda} = (\mathbf{m}, \boldsymbol{\Sigma})$, given $\boldsymbol{\phi}_t$. By simplifying the joint variational objective (Eq. 5) with $\boldsymbol{\phi}_t$, we obtain the variational objective for $\boldsymbol{\lambda}$ as:

$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}_t) = \mathbb{E}_{q_{\boldsymbol{\lambda},\ \boldsymbol{\phi}_t}(\boldsymbol{\theta}, \mathbf{A})}\big[\ell(\boldsymbol{\theta}, \mathbf{A}; \mathcal{D})\big] + \tau_1 \mathrm{KL}\big[q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) || p_0(\boldsymbol{\theta})\big] + \text{const.}$$

We approximate the expectations with respect to $q_{\boldsymbol{\lambda},\ \boldsymbol{\phi}_t}(\boldsymbol{\theta}, \mathbf{A})$ at a graph sample $\mathbf{A}_t \sim q_{\boldsymbol{\phi}_t}(\mathbf{A})$, and expand the KL term:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}_t) &\approx \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\big[\ell(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})\big] + \tau_1 \mathrm{KL}\big[q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) || p_0(\boldsymbol{\theta})\big] + \text{const.} \\
&= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\big[\ell(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})\big] + \tau_1 \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\left[\log q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) - \log p_0(\boldsymbol{\theta})\right] + \text{const.} \\
&= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\big[\ell(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D}) - \tau_1 \log p_0(\boldsymbol{\theta})\big] - \tau_1 \mathcal{H}(q_{\boldsymbol{\lambda}}) + \text{const.} \\
&= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\big[\bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})\big] - \tau_1 \tfrac{1}{2} \log |\boldsymbol{\Sigma}| + \text{const.}
\end{aligned}$$

Here, we denote by $\mathcal{H}(q)$ the entropy in the third line. The last line rewrite the problem in terms of the regularized empirical loss from Eq. 1 where the regularizer is defined as $r(\boldsymbol{\theta}) = -\tau_1 \log p_0(\boldsymbol{\theta})$. We can further simplify this objective to Laplace approximation by using second-order Taylor approximation at $\mathbf{m}$ and the delta method to remove the expectation. This is shown below by using the gradient and Hessian of $\bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})$ at $\mathbf{m}$ denoted by $\mathbf{g}_{\mathbf{m}}$ and $\mathbf{H}_{\mathbf{m}}$ respectively,

$$\begin{aligned}
&\mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{\theta})}\left[\bar{\ell}(\mathbf{m}, \mathbf{A}_t; \mathcal{D}) + (\boldsymbol{\theta} - \mathbf{m})^\top \mathbf{g}_{\mathbf{m}} + (\boldsymbol{\theta} - \mathbf{m})^\top \mathbf{H}_{\mathbf{m}} (\boldsymbol{\theta} - \mathbf{m})\right] - \tau_1 \tfrac{1}{2} \log |\boldsymbol{\Sigma}| \\
&\qquad \approx \bar{\ell}(\mathbf{m}, \mathbf{A}_t; \mathcal{D}) + \mathrm{Tr}\left[\mathbf{H}_{\mathbf{m}} \boldsymbol{\Sigma}\right] - \tau_1 \tfrac{1}{2} \log |\boldsymbol{\Sigma}|.
\end{aligned}$$

The optimization with respect to $\mathbf{m}$ reduces to minimization of $\bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})$. Denoting the minimizer by $\boldsymbol{\theta}_t$, we therefore set $\mathbf{m}_t = \boldsymbol{\theta}_t$. The optimization with respect to $\boldsymbol{\Sigma}$ is available in closed form by setting $\boldsymbol{\Sigma}_t = \mathbf{H}_{\boldsymbol{\theta}_t}^{-1}$. The optimization with respect to $\boldsymbol{\lambda}$ is thus the standard Laplace approximation.

C Approximation of the Hessian

We discuss several approximations of the Hessian matrix $\mathbf{H}_{\boldsymbol{\theta}} = \nabla^2_{\boldsymbol{\theta}} \bar{\ell}(\boldsymbol{\theta}, \mathbf{A}_t; \mathcal{D})$ used in the Laplace approximation. The Hessian is a $P \times P$ matrix, where P is the number of parameters in the GNN model. For modern neural networks, P can be in the order of millions, making direct computation and storage of the Hessian infeasible. We therefore consider several approximations to make the computation tractable. Our implementation of LAGS uses the Generalized Gauss-Newton (GGN) with block-diagonal approximation of the Hessian.

C.1 Generalized Gauss-Newton (GGN)

Generalized Gauss-Newton (GGN) extends the Gauss-Newton matrix approximation to general loss function (Schraudolph, 2002; Martens, 2010) and approximates the Hessian w.r.t the parameters. The GGN matrix is given by:

$$\mathbf{H}_{\boldsymbol{\theta}} \approx \mathbf{H}_{\boldsymbol{\theta}}^{\text{GGN}} = \sum_{i=1}^{N} [\mathbf{J}_{\boldsymbol{\theta}}^{\top} \mathbf{H}_L \mathbf{J}_{\boldsymbol{\theta}}]_i + \mathbf{P}_{\boldsymbol{\theta}},$$

where $\mathbf{J}_{\boldsymbol{\theta}} \in \mathbb{R}^{C \times P}$ is the Jacobian matrix w.r.t to the parameters, and $\mathbf{H}_L$ and $\mathbf{P}_{\boldsymbol{\theta}}$ are the Hessian of the negative log -likelihood and -prior, respectively. More specifically, $\mathbf{H}_L := -\nabla^2_{\mathbf{ff}} \log p(\mathbf{y}|\mathbf{f})$ is the Hessian of the (cross-entropy) loss w.r.t to the predictions $\mathbf{f}$, therefore has dimension $C \times C$, while the Hessian of the negative log-prior $\mathbf{P}_{\boldsymbol{\theta}} := -\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathcal{M})$ has dimension $P \times P$. So the resultant GGN matrix is $P \times P$ and the complexity is $O(NPC^2 + NCP^2)$.

C.2 Empirical Fisher (EF)

The empirical Fisher (EF) (Martens & Grosse, 2015; Grosse & Salakhutdinov, 2015) estimates the Fisher information matrix using the data $\mathcal{D}$, and can be used to estimate the Hessian:

$$\mathbf{H}_{\boldsymbol{\theta}} \approx \mathbf{H}_{\boldsymbol{\theta}}^{\text{EF}} = \sum_{i=1}^{N} \mathbf{g}_{\boldsymbol{\theta},i}^{\top} \mathbf{g}_{\boldsymbol{\theta},i} + \mathbf{P}_{\boldsymbol{\theta}},$$

where $\mathbf{g}_{\boldsymbol{\theta},i} \in \mathbb{R}^{P \times 1}$ is the gradient of the log-likelihood $\mathbf{g}_{\boldsymbol{\theta},i} := \nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}_i|\boldsymbol{\theta}, \mathcal{M})$, and $\mathbf{g}_{\boldsymbol{\theta},i}^{\top} \mathbf{g}_{\boldsymbol{\theta},i} \in \mathbb{R}^{P \times P}$ is the outer product. The resultant EF matrix is $P \times P$ and the complexity is $O(NP^2)$. For various matching pairs (such as linear activation function and square error, sigmoid and cross-entropy, and softmax and negative log-likelihood), the Fisher information matrix and GGN are equivalent (Pascanu & Bengio, 2014); this further validates EF as an approximation for the Hessian (Bottou et al., 2018).

C.3 Block and diagonal approximation

While the GGN and EF approximations allow for more manageable computation of the Hessian, practically storing and computing the determinant of a $P \times P$ matrix may still be infeasible. A solution is to make block diagonal or diagonal approximations of the Hessian. Since the Hessian is typically diagonal dominant, such an approximation is reasonable. Block diagonal approximation of the Hessian is calculated through Kronecker factorization of the Fisher matrix (Martens & Grosse, 2015) or the GGN matrix (Botev et al., 2017); that is, for each layer l of the neural network: $[\mathbf{H}_{\boldsymbol{\theta}}]_l \approx \mathbb{E}[\mathbf{Q}_l] \otimes \mathbb{E}[\mathbf{G}_l]$, where $\mathbf{Q}_l \in \mathbb{R}^{d_{l-1} \times d_{l-1}}$ matrix is calculated from the input activations to the l-th layer, and $\mathbf{G}_l \in \mathbb{R}^{d_l \times d_l}$ is based on the gradient of the output (d_l corresponds to the output dimension of the l-th layer). Diagonal approximation further reduces computation by only using the diagonal entries of the Hessian.

D Experiments

We provide additional details on the experiments from Sec. 4, including statistics of datasets used, hyper-parameters, experimental setup, and training statistics.

D.1 Datasets

We summarize the datasets used in our experiments in Table 4. Citation networks (Cora, Citeseer, Pubmed) are highly homophilic, where nodes of the same class are more likely to connect, while social networks (BlogCatalog, Flickr) and Wikipedia graphs (Squirrel, Chameleon, Roman-Empire) are more heterophilic. We follow the dataset splits of Yang et al. (2016); Kipf & Welling (2017); Platonov et al. (2023), except for Table 2, where we adopt the splits from Luo et al. (2024) for fair comparison.

Table 4: Overview of the datasets used in the experiments.

Dataset	# Nodes	# Edges	# Features	# Classes	Homophily
Cora	2,708	5,278	1,433	7	0.81
Citeseer	3,327	4,552	3,703	6	0.74
Pubmed	19,717	44,324	500	3	0.80
BlogCatalog	5,196	171,743	8,189	6	0.40
Flickr	7,575	239,738	12,047	9	0.24
Squirrel	2,223	93,996	2,089	5	0.21
Chameleon	890	17,708	2,325	5	0.24
Roman-Empire	22,662	32,927	300	18	0.05

D.2 Code

Our implementation for marginal likelihood optimization uses code from Immer et al. (2022), which uses the Automatic Second-order Differentiation Library (ASDL) from Osawa et al. (2023) for the Hessian approximation. We adapt them for our use case. Our code will be made publicly available upon acceptance. All experiments can be reproduced with a consumer grade GPU.

D.3 Overview of Hyperparameters

We provide additional details on the hyperparameters specific to LAGS.

Graph learning rate γ. The learning rate for $\boldsymbol{\phi}$. While using a lower γ with more steps K allow for more stable updates, it comes at the expense of more computations.

Graph prior probabilities $\boldsymbol{\pi}_0$. The prior probabilities $\boldsymbol{\pi}_0 \in \mathbb{R}^{N \times N}$ where each entry $\boldsymbol{\pi}_{0_{i,j}} \in (0, 1)$ is the prior probability of edge (i, j) being present in the graph. Defining $\boldsymbol{\pi}_0$ can be done in many different ways, such as using kNN-graph or using the observed graph, as we have done in our implementation.

In our implementation, we define the graph prior $\boldsymbol{\pi}_0$ by setting $\boldsymbol{\pi}_{0_{i,j}} = \pi^o$ if edge (i, j) is in the observed graph $\mathbf{A}$, and $\boldsymbol{\pi}_{0_{i,j}} = \pi^u$ otherwise. We found that for GCN, for high quality homophilic graphs (e.g. Cora, Citeseer and Pubmed), it is best to set π^o to a value close to 1. While for low quality or heterophilic graphs (e.g. BlogCatalog and Flickr), it is best to set π^o to be close to 0. For GraphSAGE, however, we found that in both cases, it is best to set the observed edges to be close to 1. In the case that the kNN graph is used as the prior, we define k as the number of nearest neighbors and $\pi^k \in (0, 1)$ as the probability of the kNN edges being present.

Temperatures (τ_1, τ_2). The temperatures determine the strength of the KL-divergence terms of Eq. 5 for $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$ respectively. To reiterate Eq. 5, the loss is:

$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\lambda},\ \boldsymbol{\phi}}(\boldsymbol{\theta}, \mathbf{A})}\big[\ell(\boldsymbol{\theta}, \mathbf{A}; \mathcal{D})\big] + \tau_1 \mathrm{KL}\big[q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) || p_0(\boldsymbol{\theta})\big] + \tau_2 \mathrm{KL}\big[q_{\boldsymbol{\phi}}(\mathbf{A}) || p_0(\mathbf{A})\big],$$

where the first term is the expectation of the likelihood. Temperature τ_1 directly corresponds to the regularization of the parameters $\boldsymbol{\theta}$, while τ_2 corresponds to the regularization of the graph posterior.

Burn-in epoch B. The number of epochs before the graph posterior is updated.

Update frequency F. The graph posterior is updated every F epochs.

Steps K. The number of steps taken to update the graph posterior.

D.4 Experimental setup and hyperparameters

The hyperparameters used in our experiments (Sec. 4) were selected based on validation loss. For the burn-in epoch B, update frequency F, and number of steps K, we searched over $5, 10, 20, 40$; for the regularization weights τ_1 and τ_2, we used $0.1, 0.5, 1.0$. The graph learning rate γ was chosen as 0.1 or 0.5. Regarding the prior probabilities, we fixed $\pi^u = 10^{-5}$ and tried π^o with 0.99 and 10^{-4}. The temperature of the binary

concrete distribution (Eq. 9) was set to 0.1. We optimized $\boldsymbol{\theta}$ with Adam (Kingma & Ba, 2015) and $\boldsymbol{\phi}$ with SGD (Robbins & Monro, 1951), using softplus activations throughout.

Table 5: Hyperparameters of LAGS-GCN for Table 1.

	Cora	Citeseer	Pubmed	BlogCatalog	Flickr	Roman-Empire
γ	0.1	0.5	0.1	0.1	0.1	0.5
π^o	0.99	0.99	0.99	1×10^{-4}	1×10^{-4}	1×10^{-4}
π^u	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
τ_1	0.5	0.5	0.1	1.0	0.0	0.0
τ_2	1.0	1.0	1.0	0.5	0.5	0.5
B	10	20	20	5	20	20
F	10	10	20	5	10	20
K	10	10	40	10	10	20
η	0.01	0.01	0.01	0.01	0.01	0.1
epochs	400	400	400	200	200	800
hidden dim.	64	64	64	64	64	128
layers	2	2	2	2	2	2
w.d.	5×10^{-3}	5×10^{-2}	5×10^{-4}	5×10^{-7}	5×10^{-7}	5×10^{-5}

Table 6: Hyperparameters of LAGS-GraphSAGE for Table 1.

	Cora	Citeseer	Pubmed	BlogCatalog	Flickr	Roman-Empire
γ	0.5	0.5	0.5	0.5	0.5	0.1
π^o	0.99	0.99	0.99	0.99	0.99	0.99
π^u	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
τ_1	0.5	0.1	0.1	0.5	0.1	0.1
τ_2	1.0	1.0	0.5	1.0	0.5	0.5
B	40	20	40	10	20	20
F	40	10	20	10	20	20
K	20	10	10	20	20	20
η	0.01	0.01	0.01	0.01	0.01	0.1
epochs	400	200	200	200	200	800
hidden dim.	64	64	64	64	64	64
layers	2	2	2	2	2	2
w.d.	5×10^{-3}	5×10^{-2}	5×10^{-4}	5×10^{-7}	5×10^{-7}	5×10^{-5}

Table 7: Hyperparameters of LAGS-GCN* for Table 2.

	Cora	Citeseer	Pubmed	Squirrel	Chameleon
γ	0.5	0.5	0.1	0.5	0.5
π^o	0.99	0.99	0.99	0.99	0.99
π^u	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
τ_1	0.1	0.1	0.1	0.1	0.1
τ_2	1.0	1.0	0.1	1.0	0.5
B	10	40	5	5	10
F	10	10	5	5	10
K	5	20	10	10	10

D.5 Wilcoxon p-values

Tables 8 and 9 provide the Wilcoxon p-values for the experiments reported in Tables 1 and 2, respectively. The p-values are calculated between LAGS and the respective baselines (GCN and GraphSAGE) over 10 runs. We observe that LAGS outperforms both GCN and GraphSAGE on most datasets in Table 1 with

p-values less than 0.05, indicating statistical significance. In Table 2, LAGS-GCN* also outperforms GCN on Cora, Citeseer, and Chameleon, again with p-values below 0.05.

Table 8: Wilcoxon p-values for LAGS-GCN and LAGS-GraphSAGE results in Table 1.

Dataset	Cora	Citeseer	Pubmed	BlogCatalog	Flickr	Roman-Empire
GCN	0.0020	0.0020	0.2324	0.0020	0.0020	0.0020
GraphSAGE	0.0020	0.0371	0.1680	0.0098	0.0137	0.0039

Table 9: Wilcoxon p-values for LAGS-GCN* results in Table 2.

Dataset	Cora	Citeseer	Pubmed	Squirrel	Chameleon
GCN	0.0020	0.0098	0.8457	0.1601	0.0486

D.6 Training Statistics

We provide the mean training statistics of LAGS in Table 10. The training time is measured in seconds per iteration, and the GPU memory usage is measured in GB.

Table 10: Training statistics of LAGS-GCN

Dataset	Cora	Citeseer	Pubmed	Blogcatalog	Flickr	Roman-Empire
Training time/iter (s)	0.2675	0.4743	0.8434	2.010	4.783	1.897
GPU memory (GB)	0.6979	1.197	31.24	3.692	7.873	42.91

We also provide additional training statistics with other GSL methods (further detailed in Appendix D.7) in comparison with LAGS-GCN and LAGS-GraphSAGE in Table 11. The discrepancy in LAGS-GCN results between Tables 10 and 11 is due to the experiments being run on different hardware. The results show that LAGS-GCN and LAGS-GraphSAGE are comparable with most other GSL methods in terms of training time and GPU memory usage.

Table 11: Training statistics of LAGS and other GSL methods on Cora.

Methods	LAGS -GCN	LAGS -GraphSAGE	WSGNN	Node Former	IDGL	IDGL -Anch	Sublime	Sublime +UnGSL	Graph Glow
Training time/iter (S)	0.6185	0.3876	0.2967	0.1041	0.3507	0.0409	1.3704	1.4579	0.1159
GPU memory (GB)	0.9622	0.9714	1.844	0.6474	2.010	0.9983	0.8263	0.9744	3.223

D.7 Other GSL Methods

We provide additional details on the methods used in Table 1. Learning Discrete Structures (LDS)(Franceschi et al., 2019) uses bi-level optimization with validation data. Iterative Deep Graph Learning (IDGL)(Chen et al., 2020b) constructs graphs from cosine similarity of node embeddings with sparsity- and smoothness-inducing losses, while IDGL-Anch adopts an anchor-based variant. NodeFormer (Wu et al., 2022) is a graph transformer that applies kernelized Gumbel-Softmax for sparse attention; WSGNN (Lao et al., 2022) employs variational inference with dual branches for observed and learned graphs; SUBLIME (Liu et al., 2022) uses unsupervised bootstrapping contrastive learning with anchor structures; and unGSL (Han et al., 2025) extends a GSL, which we chose to be SUBLIME, by modeling node-level uncertainty. GraphGLOW (Zhao et al., 2023) is a transfer learning framework which trains graph learner on other graph datasets before adapting to the target dataset. For Cora, Citeseer, and Pubmed, we used Amherst41 and Johns55 from the Facebook-100 dataset (Traud et al., 2012) to pretrain, while for BlogCatalog, Flickr, and Roman-Empire, we pretrained on the remaining two datasets in each case. We follow the hyperparameters defined from the original papers and Zhou et al. (2023).

D.8 Analysis of the learned graphs in LAGS-GCN*

We analyze the learned graphs from LAGS-GCN* in Table 2, reporting their homophily, average degree, and clustering coefficient. Homophily is the fraction of edges between nodes of the same class, the average degree is the mean number of edges per node, and the average clustering coefficient is a measure of how strongly nodes tend to cluster together. The results in Table 12 show a general increase in homophily and cluster coefficient while a decrease in average degree. The increase in homophily aligns with our expectations, as GCNs typically perform better on more homophilic graphs. Similarly, the higher clustering coefficient suggests a shift in node interactions from 2-hop neighborhoods to more localized 1-hop neighborhoods, reflecting GCN's inductive bias toward shorter-range interactions. The reduction in average degree indicates that LAGS prefers to remove noisy edges than adding new edges.

Table 12: Graph properties of the learned graphs from LAGS-GCN* with the relative increase/decrease from the original graph.

Dataset	Cora*	Citeseer*	Pubmed*	Squirrel	Chameleon
homophily	(↑ .10%) 0.8061	(↑ .29%) 0.7691	(↑ .20%) 0.7740	(↑ .18%) 0.6322	(↑ .17%) 0.5530
avg. degree	(↓ .11%) 1.942	(↓ .66%) 1.646	(↓ .04%) 1.878	(↓ .01%) 6.644	(↓ .09%) 7.280
avg. cluster coeff.	(↑ .13%) 0.2338	(↑ .23%) 0.1854	(↑ .42%) 0.1419	(↑ .19%) 0.2205	(↓ .55%) 0.2883

E Graph prior

E.1 Ablation on graph prior

Using the same experimental setup described in Section 4 and identical hyperparameters (Appendix D.3), we ablate the effect of the graph prior on LAGS-GCN performance on three configurations. In all configurations, the probability of the observed edge π^o is set to 0.99 and the probability of unobserved edges π^u is set to 0. The difference between the configurations lies in the use of a k-nearest neighbor (kNN) graph as an additional prior: the first configuration does not use a kNN graph, while the second and third configurations use a kNN graph with $k = 3$ (with probability $\pi^k = 0.1$) and $k = 5$ (with probability $\pi^k = 0.01$) neighbors respectively. Results in Table 13 shows that LAGS can improve performance of GNN even with sparse priors.

Table 13: Node classification results (%) using LAGS-GCN with different graph priors.

Dataset	Cora	Citeseer	Pubmed	BlogCatalog	Flickr	Roman-Empire
GCN	81.31 ± 0.46	71.19 ± 0.51	78.62 ± 0.42	75.48 ± 0.37	63.71 ± 0.37	52.52 ± 0.38
+ LAGS case 1	82.47 ± 0.24	72.94 ± 1.47	79.06 ± 0.27	75.61 ± 0.38	64.21 ± 0.24	54.14 ± 0.43
+ LAGS case 2	82.48 ± 0.42	71.30 ± 2.14	78.23 ± 0.60	76.17 ± 0.38	64.98 ± 0.19	54.11 ± 0.42
+ LAGS case 3	82.62 ± 0.38	71.84 ± 1.33	79.04 ± 0.68	75.66 ± 0.60	63.85 ± 0.54	52.66 ± 0.54

E.2 Scalability

Sparse graph prior (e.g. kNN graph) can be used to make learning over large graph structure more tractable. This was demonstrated with GraphSAGE on the ogbn-arxiv dataset. The hyperparameters used are in Table 15, with k being the number of neighbors in the kNN graph prior and π^k being the prior probability of the edges in the kNN graph. The batch size is set to 1024 nodes and the total training time took 4.25 hours. The results and training statistics are shown in Table 14.

F Visualization

Given the inherent difficulty of visualizing even relatively small graphs, we present a summarized visualization (Fig. 5(a)) of the graph structure learned by LAGS-GCN on the Citeseer dataset. Nodes are grouped according to their class labels, with node size proportional to the number of nodes in each label and edge

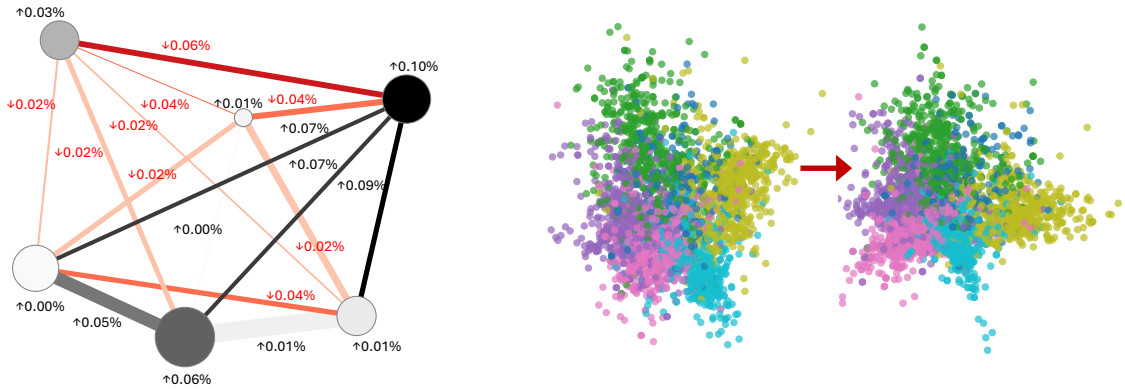
Dataset	ogbn-arxiv
GraphSAGE	62.51 ± 0.21
+LAGS	**63.46** ± 0.06
Batch size (nodes)	1024
Training time per iter (sec)	38.25
GPU memory usage (GB)	37.65

Table 14: Node classification results (%) and training statistics using the observed graph + k-nearest neighbor graph as prior.

LAGS Hyper	Value	GNN Hyper	Value
γ	0.1	η	0.01
π^o	0.99	epochs	200
π^u	0	hidden dim.	64
k	5	layers	2
π^k	1×10^{-5}	w.d.	5×10^{-5}
τ_1	0.0		
τ_2	0.5		
B	10		
F	10		
K	10		

Table 15: Hyperparameters of LAGS-GraphSAGE for ogbn-arxiv (Table 14).

width proportional to the number of edges connecting the corresponding label pairs. There is generally an increase in intra-label edges (black) and a decrease in inter-label edges (red), indicating that LAGS-GCN effectively refines the graph structure to enhance class cohesion. A projection, via principal component analysis (PCA), of the learned final node embeddings (Fig. 5(b)) between GCN and LAGS-GCN show better separation with LAGS-GCN, indicating that the learned graph structure allows it to learn a more discriminative representation of the nodes.



(a) Distribution of the learned graph using LAGS-GCN

(b) Embeddings via PCA learned by GCN (*left*) and LAGS-GCN (*right*).

Figure 5: Visualization of Citeseer. (a) Distribution of learned graph structure over labels indicates general increase (black) of intra-class edges and decrease (red) of inter-class edges. (b) Learned embeddings colored by labels shows LAGS lead to more discriminative representations.