Residual Correlation in Graph Neural Network Regression

Junteng Jia
Cornell University
jj585@cornell.edu

Austin R. Benson
Cornell University
arb@cs.cornell.edu

ABSTRACT

A graph neural network transforms features in each vertex's neighborhood into a vector representation of the vertex. Afterward, each vertex's representation is used independently for predicting its label. This standard pipeline implicitly assumes that vertex labels are conditionally independent given their neighborhood features. However, this is a strong assumption, and we show that it is far from true on many real-world graph datasets. Focusing on *regression* tasks, we find that this conditional independence assumption severely limits predictive power. This should not be that surprising, given that traditional graph-based semi-supervised learning methods such as label propagation work in the opposite fashion by explicitly modeling the correlation in predicted outcomes.

Here, we address this problem with an interpretable and efficient framework that can improve any graph neural network architecture simply by exploiting correlation structure in the regression residuals. In particular, we model the joint distribution of residuals on vertices with a parameterized multivariate Gaussian, and estimate the parameters by maximizing the marginal likelihood of the observed labels. Our framework achieves substantially higher accuracy than competing baselines, and the learned parameters can be interpreted as the strength of correlation among connected vertices. Furthermore, we develop linear time algorithms for low-variance, unbiased model parameter estimates, allowing us to scale to large networks. We also provide a basic version of our method that makes stronger assumptions on correlation structure but is painless to implement, often leading to great practical performance with minimal overhead.

ACM Reference Format:
Junteng Jia and Austin R. Benson. 2020. Residual Correlation in Graph Neural Network Regression. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403101

1 EXPLOITING RESIDUAL CORRELATION

Graphs are standard representations for wide-ranging complex systems with interacting entities, such as social networks, election maps, and transportation systems [7, 8, 22]. Typically, a graph represents entities as vertices (nodes) and the interactions as edges

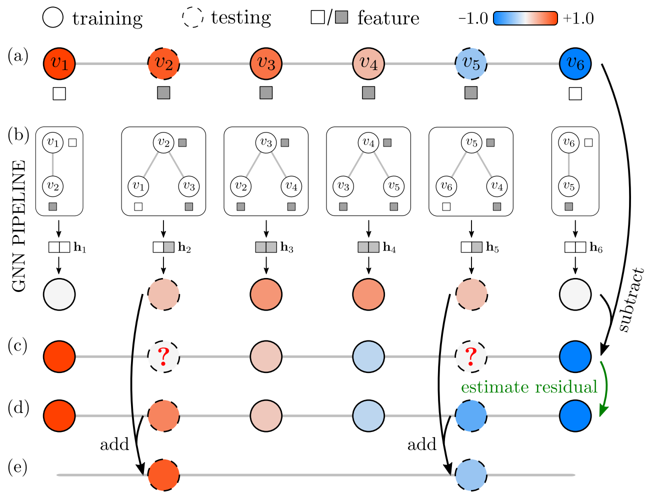

Figure 1: Limitations of GNN regression and our proposed fix. The vertex labels decrease from v_1 (+1.0) to v_6 (−1.0), and most interior vertices have positive labels. (a) Each vertex's degree is used as its feature, and vertices are colored based on their labels. The training vertices are v_1, v_3, v_4, v_6. (b) The GNN encodes vertex neighborhoods by vectors $\mathbf{h}_i$, which are used independently for label prediction. The GNN captures the positive trend for interior vertices but fails to distinguish v_1, v_2, v_3 from v_6, v_5, v_4 due to graph symmetry. (c) GNN regression residuals for the training vertices. (d) Our *Correlated GNN* method estimates the residuals on testing vertices v_2, v_5. (e) The estimated residuals are added to GNN outputs as our final predictions, yielding good estimates.

that connect two vertices. An attributed graph further records attributes of interest for each vertex; for example, an online social network may have information on a person's location, gender, and age. However, some attribute information might be missing on a subset of vertices. Continuing our online social network example, some users may skip gender during survey or registration, which one may want to infer for better targeted advertising. Or, in U.S. election map networks, we may have polling data from some counties and wish to predict outcomes in other ones, given commonly available demographic information for all the counties.

These problems fall under the umbrella of semi-supervised learning for graph-structured data. In the standard setup, one attribute (label) is observed only on a subset of vertices, and the goal is to predict missing labels using the graph topology, observed labels, and other observed attributes (features). Graph neural networks (GNNs) are a class of methods that have had great success on such tasks [13, 20, 29, 36], largely due to their ability to extract information from vertex features. The basic idea of GNNs is to first encode the local environment of each vertex into a vector representation by transforming and aggregating its own features along with the

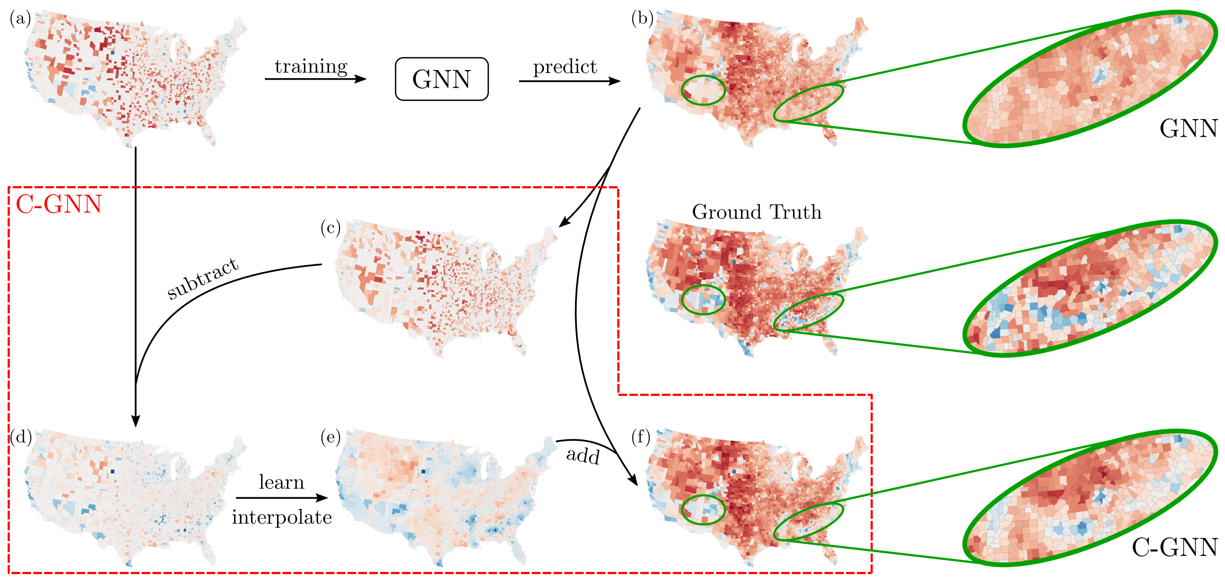

Figure 2: Our Correlated GNN (C-GNN) framework for predicting county-level outcomes in the 2016 U.S. presidential election. (a) The inputs are the county adjacency matrix, county-level demographic features, and 30% of the labels. (b) The GNN makes base predictions. (c–d) The GNN predictions on the training data (c) show that the regression residual (d) is correlated amongst neighboring counties. (e) Our C-GNN model learns the residual correlation and interpolates to get the residual on testing counties. (f) Adding predicted residuals on the test counties to the GNN base prediction substantially increases accuracy.

features of its neighbors in the graph [14], with the label prediction at a node made from its vector representation. Many target applications are for classification.[1] In this paper, we focus on regression problems. For example, in our U.S. election example above, a candidate might want to predict their vote share in each county to plan a campaign strategy. Existing GNN architectures can easily be adapted for regression problems by simply changing the output layer and choosing a loss function such as the squared error in the predicted value; automatic differentiation handles the rest.

However, a fundamental limitation of GNNs is that they predict each vertex label independently given the set of vertex representation and ignore label correlation of neighboring vertices. Specifically, a prediction depends on the features of a vertex and other vertices in its neighborhood but not on the predictions of neighboring vertices. While not stated in these terms, similar observations have been made about such limitations of GNNs [23, 32, 34]. Figures 1a and 1b illustrates why this is problematic, using a graph with topological and feature symmetry but monotonically varying vertex labels. In this example, a GNN fails to distinguish vertex v_2 from v_4 and therefore cannot predict correct labels for both of them. On the other hand, traditional graph-based semi-supervised learning algorithms (e.g., those based on label propagation [35, 37]), work very well in this case as the labels vary smoothly over the graph. Of course, in many applications, vertex features are remarkably informative. Still, gains in performance on benchmark tasks from using vertex features have in some sense put blinders on the modeler — the methodological focus is on squeezing more and more information from the features [36], ignoring signal in the joint distribution of the outcome.

[1]Perhaps the most well-studied problem in this space is predicting the “type” of an academic paper in a citation network.

In Fig. 1, vertex features partially explain the outcomes. The features are somewhat — but not overwhelmingly — predictive. The question then arises: when features are only somewhat predictive, can we get bigger gains in predictive power by exploiting outcome correlations, rather than squeezing minuscule additional signal in features with more complicated architectures?

The present work: Correlated Graph Neural Networks. To answer the above question in the affirmative, we propose *Correlated Graph Neural Networks* (C-GNNs). The basic idea of C-GNNs is to use a GNN as a base regressor to capture the (possibly mild) outcome dependency on vertex features and then further model the regression residuals on all vertices (Figs. 1c to 1e). While one can model the residual in many ways, we use a simple multivariate Gaussian with a sparse precision matrix based on the graph topology. At training, we learn the correlation structure by maximizing the marginal likelihood of the observed vertex labels. At inference, we predict the outcomes on testing vertices by maximizing their probability conditioned on the training labels. Importantly, our method covers the original GNN as a special case: minimizing squared-error loss with respect to the GNN is the maximum likelihood estimator when the precision matrix is the identity (errors are uncorrelated). We also make no assumption on the GNN architecture, as our methodology “sits on top” of the base regressor.

For a real-world data example, we predict the county-level margin of victory in the 2016 U.S. presidential election (Fig. 2). Each county is represented by a vertex with demographic features such as median household income, education levels, and unemployment rates, and edges connect bordering counties. While the GNN captures correlation between vertex features and outcomes (Figs. 2a and 2b), our C-GNN leverages residual correlation (Figs. 2c to 2f) to boost test R^2 from 0.45 to 0.63. The green circles show regions where the GNN produces large errors that are corrected by C-GNN.

More generally, we can replace the GNN base regressor with any feature-based predictor, e.g., a linear model or multilayer perceptron, and our regression pipeline is unchanged. With a linear model, for example, our framework is essentially performing generalized least squares [26], where the precision matrix structure is given by the graph. In practice, we find that within our framework, a GNN base regressor indeed works well for graph-structured data.

Our C-GNN consistently outperforms the base GNN and other competing baselines by large margins: C-GNNs achieves a mean 14% improvement in R^2 over GNNs for our datasets. Furthermore, using a simple multilayer perceptron (that does not use neighborhood features) as the base regressor, our framework even outperforms a standard GNN in most experiments. This highlights the importance of outcome correlation and suggests that focusing on minor GNN architecture improvements may not always be the right strategy.

Thus far, we have considered transductive learning, but another standard setup for machine learning on graphs is *inductive learning*: a model is trained on one graph where labels are widely available and deployed on other graphs where labels are more difficult to obtain. Assuming that the learned GNN and the residual correlation generalize to unseen graphs, our framework can simply condition on labeled vertices (if available) in a new graph to improve regression accuracy. Indeed, these assumptions hold for many real-world datasets that we consider. With a small fraction of labels in the new graphs, inductive accuracies of our C-GNN are even better than transductive accuracies of a GNN. For example, we train a model to predict county-level unemployment rates using 60% of labeled vertices in 2012. Given 10% of labels in the 2016 data, C-GNN achieves 0.65 test R^2 on unlabeled vertices, which is even more accurate than GNN trained directly on 60% of 2016 labels ($R^2 = 0.53$).

We also develop efficient numerical techniques that make model optimization tractable. Standard factorization-based algorithms for the log marginal likelihood and derivative computations require $\mathcal{O}(n^3)$ operations, where n is the number of nodes; such approaches do not scale beyond graphs with a few thousand vertices. To remedy this, we use stochastic estimation [12, 28] to take full advantage of our sparse and well-conditioned precision matrix, which reduces the computational scaling to $\mathcal{O}(m)$, where m is the number of edges, producing low-variance unbiased estimates of model parameters. We further introduce a simplified version of our method that assumes positive correlation among neighboring residuals, which is common in real-world data. The algorithm is extremely simple: train a standard GNN and then run label propagation to interpolate GNN residuals on the testing vertices. We call this LP-GNN and find that it also outperforms standard GNNs by a wide margin on a variety of real-world datasets.

2 MODELING RESIDUAL CORRELATION

Let $G = (V, E, \{\mathbf{x}_i\})$ be a graph, where V is the vertex set ($n = |V|$), E is the edge set ($m = |E|$), and $\mathbf{x}_i$ denotes the features for vertex $i \in V$. We consider the semi-supervised vertex label regression problem: given real-valued labels[2] y_L on a subset of vertices $L \subseteq V$, predict labels on the rest of the vertices $U \equiv V \backslash L$. In this section, we first review GNNs and discuss its implicit statistical assumptions. As we show in Section 4, these assumption are often invalid for real-world graph data. Motivated by this insight, we improve the predictive power of GNNs by explicitly modeling label correlations with a multivariate Gaussian, and introduce efficient numerical techniques for learning model parameters.

2.1 Statistical Interpretation of Standard GNNs

In a standard GNN regression pipeline, the features in the neighborhood of a vertex get encoded into a vertex representation,[3] and each vertex representation is used independently for label prediction:

$$\mathbf{h}_i = f\left(\mathbf{x}_i, \{\mathbf{x}_j : j \in N_K(i)\}, \theta\right); \qquad \hat{y}_i = g(\mathbf{h}_i, \theta). \tag{1}$$

Here, $N_K(i)$ denotes the K-hop neighborhood of vertex i. Oftentimes, $K = 2$ [13, 20]. The GNN weights θ are trained using observed labels, and the most common loss for regression is the squared error:

$$\textstyle\sum_{i \in L} (g(\mathbf{h}_i, \theta) - y_i)^2. \tag{2}$$

Following statistical arguments for ordinary least squares [10], minimizing Eq. (2) is equivalent to maximizing the likelihood of a factorizable joint distribution of labels, where the distribution of each label conditioned on the vertex representation is a univariate Gaussian:

$$p(\mathbf{y} \mid G) = \prod_{i \in V} p(y_i \mid \mathbf{h}_i); \quad y_i \mid \mathbf{h}_i \sim \mathcal{N}(\hat{y}_i, \sigma^2) \tag{3}$$

Consequently, the errors in the estimates $y_i - \hat{y}_i$ are i.i.d. with mean zero. However, there's no reason to assume independence, and in cases such as election data, accounting for error correlation is critical.[4] We thus consider correlation structure next.

2.2 Correlation as a Multivariate Gaussian

We model the joint distribution of labels as a multivariate Gaussian:

$$\mathbf{y} \sim \mathcal{N}\left(\hat{\mathbf{y}}, \Gamma^{-1}\right) \text{ or equivalently, } \mathbf{r} \equiv \mathbf{y} - \hat{\mathbf{y}} \sim \mathcal{N}\left(0, \Gamma^{-1}\right), \tag{4}$$

where $\Gamma = \Sigma^{-1}$ is the inverse covariance (or precision) matrix, and $\mathbf{r}$ is the residual of GNN regression. Here, we parameterize the precision matrix in a way that (i) uses the graph topology and (ii) will be computationally tractable:

$$\Gamma = \beta(\mathbf{I} - \alpha \mathbf{S}), \tag{5}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is the normalized adjacency matrix. The scalar β controls the overall magnitude of the residual, and the scalar α reflects the correlation structure. The sign of α is the direction of correlation (positive or negative), and the magnitude measures the strength of correlation.

Validity of the multivariate Gaussian requires that Γ is positive definite. This requirement is easily satisfied by restricting $-1 <$

[2]Since we are performing regression, "labels" might also be called "outcomes", or "targets"; we use those terms interchangeably in this paper.

[3]For instance, a K-step graph convolution network (GCN) computes vertex representations by repeated local feature averaging, transformation, and nonlinear activation:

$$\mathbf{h}_i^{(0)} = \mathbf{x}_i; \;\; \mathbf{h}_i^{(k)} = \phi\left(\mathbf{W}^{(k)} \cdot \text{MEAN}\left(\{\mathbf{h}_i^{(k-1)}\} \cup \{\mathbf{h}_j^{(k-1)} : j \in N_1(i)\}\right)\right); \;\; \mathbf{h}_i = \mathbf{h}_i^{(K)},$$

where $\mathbf{W}^{(k)}$ is a weight matrix at step k, and ϕ is a nonlinear activation function.

[4]https://fivethirtyeight.com/features/a-users-guide-to-fivethirtyeights-2016-general-election-forecast/

Algorithm 1: C-GNN label inference.

```
Input : normalized adjacency matrix S; features $\{\mathbf{x}_i\}$;
        training labels $\mathbf{y}_L$; parameters $\alpha, \beta$; GNN weights $\theta$
Output: predicted labels $\mathbf{y}_U^{\text{C-GNN}}$ for unknown vertices
1 $\Gamma \leftarrow \beta(\mathbf{I} - \alpha \mathbf{S})$                         ▷ precision matrix
2 $\mathbf{h}_i \leftarrow f(\mathbf{x}_i, \{\mathbf{x}_j : j \in N_K(i)\}, \theta),\ \forall i \in V$     ▷ GNN learning
3 $\hat{y}_i \leftarrow g(\mathbf{h}_i, \theta),\ \forall i \in V$                   ▷ GNN predictions
4 $\mathbf{r}_L \leftarrow \mathbf{y}_L - \hat{\mathbf{y}}_L$                       ▷ training residuals
5 $\mathbf{y}_U^{\text{C-GNN}} \leftarrow \hat{\mathbf{y}}_U - \Gamma_{UU}^{-1}\Gamma_{UL}\mathbf{r}_L$        ▷ C-GNN predictions
```

$\alpha < 1$ and $\beta > 0$. First, we verify both $(\mathbf{I}+\mathbf{S})$ and $(\mathbf{I}-\mathbf{S})$ are positive semi-definite by expanding their quadratic form with any $\mathbf{z} \in \mathbb{R}^n$:

$$\mathbf{z}^{\mathsf{T}}(\mathbf{I}+\mathbf{S})\mathbf{z} = \textstyle\sum_{(i,j)\in E}\left(z_i/\sqrt{D_{ii}} + z_j/\sqrt{D_{jj}}\right)^2 \geq 0 \tag{6}$$

$$\mathbf{z}^{\mathsf{T}}(\mathbf{I}-\mathbf{S})\mathbf{z} = \textstyle\sum_{(i,j)\in E}\left(z_i/\sqrt{D_{ii}} - z_j/\sqrt{D_{jj}}\right)^2 \geq 0 \tag{7}$$

For $0 \leq \alpha < 1$, $\Gamma = (1-\alpha)\beta\mathbf{I} + \alpha\beta(\mathbf{I}-\mathbf{S}) \succ 0$ since the first term is strictly positive definite, and the second term is positive semi-definite. A similarly argument holds for $-1 < \alpha < 0$. Two special cases of the precision matrix in Eq. (5) deserve special attention. First, when $\alpha = 0$, Γ is the identity matrix (up to constant scaling), and the model reduces to standard GNN regression. Second, in the limit $\alpha \to 1$, Γ is the normalized Laplacian matrix, and the noise is assumed to be smooth over the entire graph. The normalized Laplacian matrix is only positive semi-definite, so we make sure the limit is never realized in practice; however, we use this as motivation for a simplified version of the model in Section 2.3.

Inferring unknown labels. Now we show how to infer unlabeled vertices assuming α, β, θ, and y_L are known. If we partition Eq. (4) into the labeled and unlabeled blocks,

$$\begin{bmatrix}\mathbf{y}_L\\ \mathbf{y}_U\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\hat{\mathbf{y}}_L\\ \hat{\mathbf{y}}_U\end{bmatrix}, \begin{bmatrix}\Gamma_{LL} & \Gamma_{LU}\\ \Gamma_{UL} & \Gamma_{UU}\end{bmatrix}^{-1}\right), \tag{8}$$

then conditioned on the labeled vertices L, the distribution of vertex labels on U is also a multivariate Gaussian,

$$\mathbf{y}_U \mid \mathbf{y}_L \sim \mathcal{N}\left(\hat{\mathbf{y}}_U - \Gamma_{UU}^{-1}\Gamma_{UL}\mathbf{r}_L, \Gamma_{UU}^{-1}\right). \tag{9}$$

Our model uses the expectation of this conditional distribution as the final prediction, which is given by the Gaussian mean,

$$\mathbf{y}_U^{\text{C-GNN}} = \hat{\mathbf{y}}_U - \Gamma_{UU}^{-1}\Gamma_{UL}\mathbf{r}_L. \tag{10}$$

Algorithm 1 summarizes the inference algorithm. Next, we consider learning optimal parameters from labeled data.

Learning model parameters. Given the observed outcomes y_L on L, the precision matrix parameters α, β and GNN weights θ are learned by maximum likelihood estimation. The marginal distribution of the GNN residual on L is a multivariate Gaussian [24]:

$$\mathbf{r}_L = \mathbf{y}_L - \hat{\mathbf{y}}_L \sim \mathcal{N}\left(0, \bar{\Gamma}_{LL}^{-1}\right), \tag{11}$$

where $\bar{\Gamma}_{LL} = \Gamma_{LL} - \Gamma_{LU}\Gamma_{UU}^{-1}\Gamma_{UL}$ is the corresponding precision matrix. We define the loss function as the negative log marginal likelihood of observed labels:

Algorithm 2: C-GNN training (mini-batched).

```
Input : normalized adjacency matrix S; features $\{\mathbf{x}_i\}$; all
        training vertices $L_0$, labels $\mathbf{y}_{L_0}$; number of training
        steps $p$; batch size $b$
Output: optimized $\alpha, \beta, \theta$
1  randomly initialize $\alpha, \beta, \theta$
2  for $i \leftarrow 1$ to $p$ do
3      $\Gamma \leftarrow \beta(\mathbf{I} - \alpha\mathbf{S})$
4      $L \leftarrow$ subsample($L_0$, $b$)                              ▷ get mini-batch
5      $\mathbf{h}_i \leftarrow f(\mathbf{x}_i, \{\mathbf{x}_j : j \in N_K(i)\}, \theta),\ \forall i \in L$
6      $\hat{y}_i \leftarrow g(\mathbf{h}_i, \theta),\ \forall i \in L$                    ▷ GNN predictions
7      $\mathbf{r}_L \leftarrow \mathbf{y}_L - \hat{\mathbf{y}}_L$                        ▷ training residuals
8      $\Omega \leftarrow \mathbf{r}_L^{\mathsf{T}}\bar{\Gamma}_{LL}\mathbf{r}_L - \log\det(\Gamma) + \log\det(\Gamma_{UU})$
9      compute $\partial\Omega/\partial\alpha, \partial\Omega/\partial\beta, \partial\Omega/\partial\theta$                 ▷ Eq. (13)
10     $\alpha, \beta, \theta \leftarrow$ update($\Omega, \partial\Omega/\partial\alpha, \partial\Omega/\partial\beta, \partial\Omega/\partial\theta$)
11 end
```

$$\begin{aligned}\Omega &= -\log p(\mathbf{y}_L|\alpha,\beta,\theta)\\ &= \left[\mathbf{r}_L^{\mathsf{T}}\bar{\Gamma}_{LL}\mathbf{r}_L - \log\det(\bar{\Gamma}_{LL}) + n\log(2\pi)\right]/2\\ &\propto \mathbf{r}_L^{\mathsf{T}}\bar{\Gamma}_{LL}\mathbf{r}_L - \log\det(\Gamma) + \log\det(\Gamma_{UU})\end{aligned} \tag{12}$$

Then, the loss function derivatives with respect to the model parameters take the following expression,

$$\begin{aligned}\frac{\partial\Omega}{\partial\alpha} &= \mathbf{r}_L^{\mathsf{T}}\frac{\partial\bar{\Gamma}_{LL}}{\partial\alpha}\mathbf{r}_L - \mathtt{tr}\left(\Gamma^{-1}\frac{\partial\Gamma}{\partial\alpha}\right) + \mathtt{tr}\left(\Gamma_{UU}^{-1}\frac{\partial\Gamma_{UU}}{\partial\alpha}\right)\\ \frac{\partial\Omega}{\partial\beta} &= \mathbf{r}_L^{\mathsf{T}}\frac{\partial\bar{\Gamma}_{LL}}{\partial\beta}\mathbf{r}_L - \mathtt{tr}\left(\Gamma^{-1}\frac{\partial\Gamma}{\partial\beta}\right) + \mathtt{tr}\left(\Gamma_{UU}^{-1}\frac{\partial\Gamma_{UU}}{\partial\beta}\right)\\ \frac{\partial\Omega}{\partial\theta} &= -2\mathbf{r}_L^{\mathsf{T}}\bar{\Gamma}_{LL}\frac{\partial\hat{\mathbf{y}}_L}{\partial\theta},\end{aligned} \tag{13}$$

where $\partial\hat{\mathbf{y}}_L/\partial\theta$ can be computed with back-propagation, and

$$\begin{aligned}\frac{\partial\bar{\Gamma}_{LL}}{\partial\alpha} &= \frac{\partial\Gamma_{LL}}{\partial\alpha} - \frac{\partial\Gamma_{LU}}{\partial\alpha}\Gamma_{UU}^{-1}\Gamma_{UL} + \Gamma_{LU}\Gamma_{UU}^{-1}\frac{\partial\Gamma_{UU}}{\partial\alpha}\Gamma_{UU}^{-1}\Gamma_{UL}\\ &\quad - \Gamma_{LU}\Gamma_{UU}^{-1}\frac{\partial\Gamma_{UL}}{\partial\alpha}\\ \frac{\partial\bar{\Gamma}_{LL}}{\partial\beta} &= \frac{\partial\Gamma_{LL}}{\partial\beta} - \frac{\partial\Gamma_{LU}}{\partial\beta}\Gamma_{UU}^{-1}\Gamma_{UL} + \Gamma_{LU}\Gamma_{UU}^{-1}\frac{\partial\Gamma_{UU}}{\partial\beta}\Gamma_{UU}^{-1}\Gamma_{UL}\\ &\quad - \Gamma_{LU}\Gamma_{UU}^{-1}\frac{\partial\Gamma_{UL}}{\partial\beta}.\end{aligned} \tag{14}$$

Finally, let P, Q denote two arbitrary sets of vertices. The derivatives of each precision matrix block Γ_{PQ} are given by $\partial\Gamma_{PQ}/\partial\alpha = -\beta\mathbf{S}_{PQ}$ and $\partial\Gamma_{PQ}/\partial\beta = \Gamma_{PQ}/\beta$. In practice, we employ a mini-batch sampling for better memory efficiency, and we maximize the marginal likelihood of a mini-batch at each training step (Algorithm 2).

One remaining issue is the computational cost. Standard matrix factorization-based algorithms for computing the matrix inverse and log determinant have complexity cubic in the number of vertices, which is computationally prohibitive for graphs beyond a few thousand vertices. In Section 3, we show how to reduce these computations to linear in the number of edges, using recent tricks in stochastic trace estimation. Next, we offer an even cheaper alternative that works well when α is close to 1.

Algorithm 3: LP-GNN regression.

```
Input  : normalized adjacency matrix S; features $\{\mathbf{x}_i\}$;
         training labels $\mathbf{y}_L$
Output: predicted labels $\mathbf{y}_U^{\text{LP-GNN}}$ for unknown vertices
1 train standard GNN, get optimized parameter $\theta$
2 $\mathbf{h}_i \leftarrow f(\mathbf{x}_i, \{\mathbf{x}_j : j \in N_K(i)\}, \theta),\ \forall i \in V$
3 $\hat{y}_i \leftarrow g(\mathbf{h}_i, \theta),\ \forall i \in V$          ▷ GNN predictions
4 $\mathbf{r}_L \leftarrow \mathbf{y}_L - \hat{\mathbf{y}}_L$          ▷ training residuals
5 $\mathbf{r}_U^{\text{est}} \leftarrow \texttt{LabelPropagation}(\mathrm{S}, \mathbf{r}_L)$          ▷ e.g., Algorithm 4
6 $\mathbf{y}_U^{\text{LP-GNN}} \leftarrow \hat{\mathbf{y}}_U + \mathbf{r}_U^{\text{est}}$
```

2.3 A Simple Propagation-based Algorithm

Our framework is inspired in part by label propagation [35, 37], where the neighboring correlation is always assumed to be positive. In fact, if we fix $\alpha = 1$ and replace the base GNN regressor with one that gives uniform 0 prediction for all vertices, our method reduces to a variant of label propagation that uses the normalized Laplacian matrix (see details in Appendix A.1), which might be expected given the connection between Gaussian Process regression (kriging) and graph-based semi-supervised learning [33].

This observation motivates an extremely simple version of our method, which we call LP-GNN (Algorithm 3): (i) train a standard GNN; (ii) run label propagation from the residuals on labeled vertices; (iii) add the propagated result to the GNN predictions. LP-GNN is a lightweight framework for data where residual correlation is strong and positive, and in principle, any label propagation method could be employed. We show in Section 4 that this provides substantial improvements over a GNN in many cases, but the C-GNN still has better predictions.

2.4 Extension to Multiple Edge Types

Our model can also be extend to study graphs with multiple edge types. For instance, later in Section 4.2, we consider a road traffic network where different pairs of lanes, based on their orientations, are connected with different types of edges. In this setting, we decompose the total adjacency matrix as $\mathbf{A} = \sum_i \mathbf{A}^{(i)}$, where $\mathbf{A}^{(i)}$ is given by the edges of type i. Then, denoting $\mathrm{S}^{(i)} = \mathbf{D}^{-1/2}\mathbf{A}^{(i)}\mathbf{D}^{-1/2}$, we parametrize the precision matrix as

$$\Gamma = \beta(\mathbf{I} - \textstyle\sum_i \alpha_i \mathrm{S}^{(i)}). \tag{15}$$

Following the same logic as in Section 2.2, the above precision matrix is still positive definite if $-1 < \alpha_i < 1$ for all i, and the loss function derivatives with respect to $\{\alpha_i\}$ are similar to the original model. The extended model provides finer grained descriptions for interactions among neighboring vertices. Our experiments show that the extended model captures the difference in correlation strengths for different types of edges in the traffic network, as well as improving the regression accuracy.

3 FAST MODEL OPTIMIZATION

We have introduced a simple and interpretable framework to exploit residual correlations. However, the model's applicability to large-scale networks is limited by the cubic-scaling cost associated with the log-determinant computations during learning. Here, we use stochastic estimation of the log determinant and its derivatives. By taking advantage of our sparse precision matrix parametrization, this makes computations essentially linear in the size of the graph.

3.1 Efficient Log-determinant Estimation

The major computational cost in our framework boils down to three types of matrix computations: (i) solving the linear system $\Gamma^{-1}\mathbf{z}$; (ii) the matrix trace $\texttt{tr}(\Gamma^{-1}\frac{\partial\Gamma}{\partial\alpha})$; and (iii) the log determinant $\log\det(\Gamma)$.[5] Next, we show how our precision matrix parametrization allows those operations to be computed efficiently using conjugate gradients (CG), stochastic trace estimation, and Lanczos quadrature [2, 6, 9, 28].

Conjugate Gradients (CG) solution of $\Gamma^{-1}\mathbf{z}$. CG is an iterative algorithm for solving linear systems when the matrix is symmetric positive definite. Each CG iteration computes one matrix vector multiplication and a handful of vector operations, so approximately solving $\Gamma^{-1}\mathbf{z}$ with k CG iterations requires $O(km)$ operations, where m is the number of edges in the graph. The convergence rate of CG depends on the condition number of Γ, which is the ratio between the largest and smallest eigenvalues: $\kappa(\Gamma) = \lambda_{\max}(\Gamma)/\lambda_{\min}(\Gamma)$. In particular, for a fixed error tolerance, CG converges in $O(\sqrt{\kappa(\Gamma)})$ iterations. We now provide an upper bound on $\kappa(\Gamma)$, which justifies using a fixed number of iterations.

Since the eigenvalues of the normalized adjacency matrix S are bounded between -1.0 and 1.0 [4], we can bound the extreme eigenvalues of the precision matrix as follows:

$$\lambda_{\max}(\Gamma) = \beta\lambda_{\max}(\mathbf{I} - \alpha\mathrm{S}) < \beta[\lambda_{\max}(\mathbf{I}) + \lambda_{\max}(-\alpha\mathrm{S})] = \beta(1 + |\alpha|)$$
$$\lambda_{\min}(\Gamma) = \beta\lambda_{\min}\left[(1 - |\alpha|)\mathbf{I} + |\alpha|\left(\mathbf{I} - \frac{\alpha}{|\alpha|}\mathrm{S}\right)\right] > \beta(1 - |\alpha|) \tag{16}$$

Then, the upper bound of the condition number is

$$\kappa(\Gamma) = \lambda_{\max}(\Gamma)/\lambda_{\min}(\Gamma) < (1 + |\alpha|)/(1 - |\alpha|), \tag{17}$$

which does not depend on the graph topology. (This upper bound also applies to Γ_{UU} via the eigenvalue interlacing theorem.) Therefore, by further constraining $|\alpha| < 1 - \eta$ for a small positive constant η, CG algorithm converges in $O(\sqrt{2/\eta})$ iterations. We will verify numerically in Section 3.2 that in practice, CG converges in a couple dozen iterations for our framework.

Stochastic Estimation of $\texttt{tr}(\Gamma^{-1}\frac{\partial\Gamma}{\partial\alpha})$. The stochastic trace estimator is an established method for approximating the trace of a matrix function [2, 16, 28]. Given a Gaussian random vector $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ with $E[z_i z_j] = \delta_{ij}$, where δ_{ij} is the Kronecker delta function,

$$\mathbb{E}\left[\mathbf{z}^\intercal\mathbf{M}\mathbf{z}\right] = \mathbb{E}\left[\textstyle\sum_i z_i^2 M_{ii} + \sum_{i\neq j} z_i z_j M_{ij}\right] = \sum_i M_{ii} \tag{18}$$

gives the unbiased trace estimation for any matrix $\mathbf{M}$. This allows us to estimate $\texttt{tr}(\Gamma^{-1}\frac{\partial\Gamma}{\partial\alpha})$ without explicitly forming Γ^{-1}. In practice, given T independent and identically sampled Gaussian random vectors $\mathbf{z}_t \sim \mathcal{N}(0, \mathbf{I})$, $t = 1, \ldots, T$, we estimate the matrix trace by

$$\texttt{tr}(\Gamma^{-1}\frac{\partial\Gamma}{\partial\alpha}) = \mathbb{E}\left[\mathbf{z}_t^\intercal\Gamma^{-1}\frac{\partial\Gamma}{\partial\alpha}\mathbf{z}_t\right] \approx \frac{1}{T}\sum_{t=1}^{T}\left(\Gamma^{-1}\mathbf{z}_t\right)^\intercal\left(\frac{\partial\Gamma}{\partial\alpha}\mathbf{z}_t\right), \tag{19}$$

which would require calling the conjugate gradient solver T times with the same matrix Γ but different right-hand-sides.

[5]We focus on evaluating $\log\det(\Gamma)$ and $\partial\Omega/\partial\alpha$ in our analysis, but the results easily generalize to $\log\det(\Gamma_{UU})$ and $\partial\Omega/\partial\beta$.

Stochastic Lanczos quadrature for $\log\det(\Gamma)$**.** We adopt the approach of Ubaru et al. for approximating the log-determinant, which estimates the trace of the logarithm of the matrix [28]:

$$\begin{aligned}\log\det(\Gamma) = \mathtt{tr}(\log\Gamma) &\approx \tfrac{1}{T}\textstyle\sum_{t=1}^{T} \mathbf{z}_t^\intercal \log\Gamma \mathbf{z}_t \\ &= \tfrac{1}{T}\textstyle\sum_{t=1}^{T} \mathbf{z}_t^\intercal \mathbf{Q}\log\Lambda\mathbf{Q}^\intercal \mathbf{z}_t \\ &= \tfrac{1}{T}\textstyle\sum_{t=1}^{T}\sum_{i=1}^{n} \mu_{ti}^2\cdot\log\lambda_i(\Gamma), \qquad (20)\end{aligned}$$

where $\Gamma = \mathbf{Q}\Lambda\mathbf{Q}^\intercal$ is the eigen-decomposition, and μ_{ti} is the projected length of $\mathbf{z}_t$ on the i-th eigenvector of Γ. The expression $\sum_i^n \mu_{ti}^2 \cdot \log\lambda_i(\Gamma)$ can be considered as a Riemann-Stieltjes integral, and is further approximated with Gaussian quadrature:

$$\textstyle\sum_{i=1}^{n} \mu_{ti}^2\cdot\log\lambda_i(\Gamma) \approx \sum_{i=1}^{k} w_{ti}^2\cdot\log\xi_{ti}, \qquad (21)$$

where the optimal nodes $\{\xi_{ti}\}$ and weights $\{w_{ti}\}$ for numerical integration are determined as follows. First, run k steps of the Lanczos algorithm with Γ and initial vector $\mathbf{z}_t$ to get $\mathbf{V}_t^\intercal\Gamma\mathbf{V}_t = \mathbf{T}_t$. Then, perform the eigen-decomposition of the tri-diagonal matrix $\mathbf{T}_t = \mathbf{P}_t\Xi_t\mathbf{P}_t^\intercal$. Each integration node is an eigenvalue of $\mathbf{T}_t$ whose weight is the first element of each corresponding eigenvector:

$$\xi_{ti} = (\Xi_t)_{ii}, \qquad w_{ti} = \sqrt{n}\cdot(\mathbf{P}_t)_{1i} \qquad (22)$$

Please see Ubaru et al. for a complete derivation [28].

Implementation and algorithm complexity. Both the CG and Lanczos algorithms are Krylov subspace methods, and their convergence essentially depends on the condition number [30]. Since the condition number in our precision matrix parametrization is bounded, we use a fixed number of k iterations in both algorithms. Furthermore, the error of the stochastic trace estimator decrease with the number of trial vectors T as $\mathbf{O}(T^{-1/2})$, regardless of the graph topology, and we also use a fixed number of T vectors.

We summarize the overall complexity of the proposed method for evaluating Eqs. (12) and (13) in each optimization step. Computing $\hat{\mathbf{y}}_L$ and $^{\partial\hat{\mathbf{y}}_L}/_{\partial\theta}$ through forward and back propagation takes $\mathbf{O}(n)$ operations (assuming constant-size neighborhood subsampling in the GNN implementation). Evaluating the quadratic forms in Eq. (13) invokes a constant number of calls (8 in our case) to the CG solver, which takes $\mathbf{O}(mk)$ operations. The trace estimations $\mathtt{tr}(\Gamma^{-1}\frac{\partial\Gamma}{\partial\alpha})$ invokes T calls to the CG solver, which takes $\mathbf{O}(mkT)$ operations. The log-determinant estimation $\log\det(\Gamma)$ invokes T calls to the Lanczos algorithm, which takes $\mathbf{O}(mkT)$ operations. Finally, the eigen-decomposition of the k-by-k tri-diagonal matrices $\{\mathbf{T}_t\}_{t=1}^{T}$ takes $O(Tk^2)$ operations. We choose $T = 128, k = 32$ as the default hyperparameters, independent of the size of the graph for an overall complexity of $O(m)$, i.e., linear in the number of edges.

Stochastic estimation of the log determinant and its derivatives of the covariance matrix has been considered in the context of Gaussian Processes [12], where a similar computational scheme is used to reduce the complexity from $O(n^3)$ to $O(n^2)$. Our model further benefits from the sparse and well-conditioned precision matrix parametrization, which results in linear-time computations of the objective function and its gradients. We implement the log-determinant estimation function in Julia using the `Flux.jl` automatic differentiation package [18], which automatically tracks the function value and derivatives (see Appendix A.2 for details). We also adapt techniques proposed by Gardner et al. for reusing computation and improving cache efficiency [12].

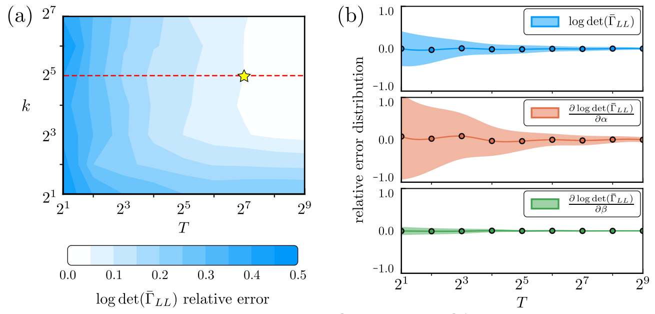

Figure 3: Estimation error as a function of hyperparameters. (a) Relative error of log determinant estimation. The yellow star marks our default hyperparameters. (b) Relative error distribution along the red dashed line in (a) of the log determinant and its derivatives as a function of T.

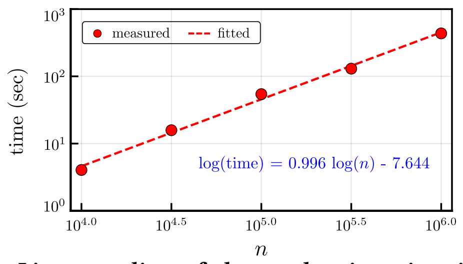

Figure 4: Linear scaling of the stochastic estimation algorithm using random Watts-Strogatz graphs, where the average degree in each graph is 10. Measured times are circles, and the dashed line is the linear fit (coefficients in blue).

3.2 Validation of Stochastic Estimation

We now demonstrate the accuracy of the proposed stochastic estimation algorithm as a function of the hyperparameters T and k. We find that the proposed scheme gives accurate and unbiased estimates for the log determinant and its derivatives for modest values of T and k, and we empirically show linear scaling.

Accuracy in estimating log determinant and its derivatives. To test our fast algorithms, we sample a Watts-Strogatz graph [31] with 500 vertices and average vertex degree 10. We randomly select 50% vertices as labeled, and compute the marginal precision matrix $\bar{\Gamma}_{LL}$ with $\alpha = 0.999$ and $\beta = 1.0$, which corresponds to an ill-conditioned parametrization. To understand how the quality of the approximation depends on the hyperparameters, we compare our stochastic algorithm output to "ground truth" log-determinant and derivatives obtained from factorization-based algorithms. The estimation accuracy is measured by the root mean square relative error over 100 runs (for various T, k; Fig. 3). Under the default hyperparameters ($T = 128, k = 32$), the relative error between log-determinant estimation and the ground truth is less than 5%. Moreover, our algorithm produces unbiased estimates for the derivatives with respect to α and β, which enables us to confidently use stochastic gradient methods for learning those parameters.

Scalability of stochastic estimation. Now, we validate the computational complexity of the proposed algorithm. We run our algorithm on a sequence of Watts-Strogatz graphs with increasing number of vertices, with average degree fixed to be 10. Figure 4 shows that the empirical running time grows linearly with the size of the graph, as indicated by the slope of the fitted curve.

4 NUMERICAL EXPERIMENTS

Now that we have developed efficient algorithms for optimizing model parameters, we use our model for label prediction tasks in synthetic and real-world attributed graphs. Our model learns both positive and negative residual correlations from real-world data, which substantially boosts the regression accuracy and also provides insights about the correlation among neighboring vertices.

4.1 Data

Our model and the baselines are tested on the following graphs (see Appendix A.4 for additional datasets details).

Ising model. The Ising model is a widely-used random model in statistical physics [5], and we consider vertices on a 35×35 grid graph. The spin of each vertex is either up ($+1.0$) or down (-1.0), which tends to align with an external field but is also influenced by neighboring spins. The neighboring spins are likely to be parallel if their interaction is set to be positive, and anti-parallel otherwise. We use Ising model samples from these two settings and denote them by Ising(+) and Ising(-), respectively, providing synthetic datasets with clear positive and negative correlations in labels. We use the grid coordinates as vertex features to predict vertex spins.

U.S. election maps. The election map is in Fig. 2, where vertices are counties in the U.S. and edges connect bordering counties. Each county has demographic and election statistics.[6] We use these as both features and outcomes: in each experiment, we select one statistic as the outcome; the remaining are vertex features. We use 2012 and 2016 statistics. The former is used for the transductive experiments, and both are used for the inductive experiments.

Transportation networks. The transportation networks contain traffic data in the cities of Anaheim and Chicago.[7] Each vertex represents a directed lane, and two lanes that meets at the same intersection are connected. Since the lanes are directed, we create two type of edges: lanes that meets head-to-tail are connected by a type-1 edge, and lanes that meet head-to-head or tail-to-tail are connected by a type-2 edge. For this, we use the extended model from Section 2.4. The length, capacity, and speed limit of each lanes are used as features to predict traffic flows on the lanes.

Sexual interactions. The sexual interaction network among 1,888 individuals is from an HIV transmission study [21]. We use race and sexual orientation as vertex features to predict the gender of each person ($+1$ for male / -1 for female). Most sexual interactions are heterosexual, producing negative label correlations for neighbors.

Twitch social network. The Twitch dataset represents online friendships amongst Twitch streamers in Portugal [25]. Vertex features are principal components from statistics such as the games played and liked, location, and streaming habits. The goal is to predict the logarithm of the number of viewers for each streamer.

4.2 Transductive Learning

We first consider the transductive setting, where the training and testing vertices are from the same graph. We find that our C-GNN framework greatly improves prediction accuracy over GNNs.

[6]Graph topology and election outcomes from https://github.com/tonmcg/, other statistics from www.ers.usda.gov/data-products/county-level-data-sets/.
[7]Data from https://github.com/bstabler/TransportationNetworks.

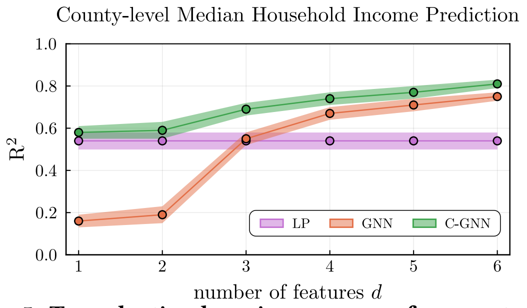

Figure 5: Transductive learning accuracy for county-level median household incomes, as a function of the number of included features. Label propagation (LP) can work well with few features, while the GNN can work well with many features; however, C-GNN outperforms both in all cases.

Methods and baselines. We use a 2-layer GraphSAGE GNN with ReLU activations and mean-aggregation [13] as the base predictor in our framework. (Other GNN architecture provide similar results; see Appendix A.5.) We compare C-GNN against label propagation (LP) [37], a multi-layer perceptron (MLP; architecture details in Appendix A.3), the base GNN, and the LP-GNN algorithm from Section 2.3. LP assumes and takes advantage of positive label correlation among neighboring vertices, but it does not use vertex features. On the other hand, the MLP ignores the label correlations, and only uses the features of a given vertex to predict its label. We also tested our correlation framework with the MLP as the base regressor instead of the GNN (C-MLP and LP-MLP).

Setup and performance metric. For each graph, we normalize each vertex feature to have zero mean and unit standard deviation, and randomly split the vertices into 60% for training, 20% for validation, and 20% for testing. The GNN parameters are trained using the ADAM optimizer with default learning rate, while the model parameters α, β are optimized with gradient descent along with the GNN parameters. For the Ising model and sexual interaction datasets, the vertex labels are binary, so we threshold the regression output at 0 and use binary classification accuracy as the performance metric. For the rest of datasets, we use coefficients of determination R^2 to measure accuracy. Each combination of method and dataset is repeated 10 times with different random seeds, and the mean and standard deviation of the accuracies are recorded.

Main results. Table 1 summarizes the results. C-GNN substantially improves the prediction accuracy over GNN for all datasets: the C-GNN mean classification accuracy is 0.82 over the Ising spin and sexual interaction datasets, and the mean R^2 is 0.75 over the remaining datasets, while the GNN mean classification and R^2 accuracies were 0.67 and 0.66, respectively. Moreover, our LP-GNN also performs very well on most of the datasets, with performance on par with C-GNN in five datasets and performing at least as well as the standard GNN in 8 out of 10 datasets. The two datasets on which it performs poorly are Ising(-) and the sexual interaction network, where the labels of connected vertices are likely to be negatively correlated; this is expected since the LP-GNN model assumes positive correlations between neighbors. Interestingly, our framework also significantly improves the performance of the MLP. In fact, C-MLP is often much better than a standard GNN. This is evidence that oftentimes more performance can be gained

Table 1: Transductive learning accuracy of our C-GNN and LP-GNN models compared to competing baselines. The best accuracy is in green. Our C-GNN outperforms GNN on all datasets, often by a substantial margin. Even C-MLP, which does not use neighbor features, outperforms GNN in many cases, highlighting the importance of label correlation. LP, LP-MLP and LP-GNN assume positive label correlation among neighboring vertices and perform poorly for datasets where most edges encode negative interactions, as highlighted in orange. We also report the learned $\{\alpha_i\}$ values from C-GNN.

Dataset	n	m	LP	MLP	LP-MLP	C-MLP	GNN	LP-GNN	C-GNN	$\{\alpha_i\}$
Ising(+)	1.2K	2.4K	**0.76** ± 0.02	0.68 ± 0.03	**0.76** ± 0.02	**0.76** ± 0.02	0.67 ± 0.04	**0.76** ± 0.02	**0.76** ± 0.02	+0.89
Ising(-)	1.2K	2.4K	**0.30** ± 0.03	0.47 ± 0.02	**0.30** ± 0.03	**0.77** ± 0.03	0.47 ± 0.03	**0.30** ± 0.03	**0.77** ± 0.03	−0.93
income	3.2K	12.7K	0.54 ± 0.04	0.64 ± 0.03	0.73 ± 0.03	0.74 ± 0.03	0.75 ± 0.03	**0.81** ± 0.03	**0.81** ± 0.02	+0.92
education	3.2K	12.7K	0.36 ± 0.05	0.67 ± 0.03	0.71 ± 0.02	**0.72** ± 0.02	0.70 ± 0.02	**0.72** ± 0.03	**0.72** ± 0.03	+0.78
unemployment	3.2K	12.7K	0.70 ± 0.03	0.43 ± 0.05	0.69 ± 0.04	0.77 ± 0.03	0.55 ± 0.04	0.75 ± 0.05	**0.78** ± 0.03	+0.99
election	3.2K	12.7K	0.58 ± 0.02	0.37 ± 0.02	0.61 ± 0.03	0.63 ± 0.03	0.51 ± 0.04	**0.69** ± 0.03	**0.69** ± 0.03	+0.95
Anaheim	914	3.8K	0.49 ± 0.08	0.75 ± 0.02	0.81 ± 0.04	**0.82** ± 0.03	0.76 ± 0.03	0.81 ± 0.04	**0.82** ± 0.03	+0.95, +0.17
Chicago	2.2K	15.1K	0.59 ± 0.05	0.60 ± 0.05	0.65 ± 0.06	0.65 ± 0.05	0.68 ± 0.04	**0.72** ± 0.04	0.71 ± 0.04	+0.85, +0.68
sexual	1.9K	2.1K	**0.37** ± 0.06	0.68 ± 0.02	**0.64** ± 0.03	0.83 ± 0.03	0.88 ± 0.02	**0.86** ± 0.02	**0.93** ± 0.01	−0.98
Twitch-PT	1.9K	31.3K	0.00 ± 0.04	0.61 ± 0.03	0.60 ± 0.04	0.66 ± 0.03	0.69 ± 0.03	0.69 ± 0.03	**0.74** ± 0.03	+0.99

from modeling label correlation as opposed to sophisticated feature aggregation.

The learned parameters also reveal interaction types. The learned $\{\alpha_i\}$ are all positive except for the Ising(-) and sexual interaction datasets, where the vertex labels are negatively correlated. Moreover, for the traffic graph, the learned $\alpha_1 > \alpha_2$ indicates that traffic on two lanes with head-to-tail connection are more strongly correlated, since a vehicle can directly move from one lane to another.

Understanding performance better. We perform a more in-depth analysis for predicting county-level median household income. This dataset has six features (migration rate, birth rate, death rate, education level, unemployment rate, and election outcome), and we use the first d for income prediction, comparing against LP and GNN (Fig. 5). The GNN performs poorly for small d, but gradually surpasses LP as more features are available. Our C-GNN method outperforms both regardless of d, although the performance gap between C-GNN and GNN narrows as d increases. These results highlight that, if features are only mildly predictive, accounting for label correlation can have an enormous benefit.

4.3 Inductive Learning

We now consider the inductive setting, where a model is trained on vertex labels from one graph G and tested on an unseen graph G'. This setting is useful when both graphs have similar properties, but vertex labels in G' are more expensive to obtain. In particular, we consider the scenario where a small fraction of vertex labels in G' are available and demonstrate how our framework allows using those extra labels to improve regression accuracy. We denote the labeled and unlabeled vertices in G' as L' and U'.

Datasets and methods. We use the Ising model and election map datasets for inductive learning experiments. In the former, the spin configurations on G and G' are simulated under the same Ising model setting. In the latter, we train with the 2012 data and test on the 2016 election map, predicting several attributes. We compare C-GNN to GNN and MLP. The C-GNN is trained using 60% vertex labels from G, and tested directly on U' by conditioning on the vertex labels of L'. The GNN and MLP are first trained on G; for a

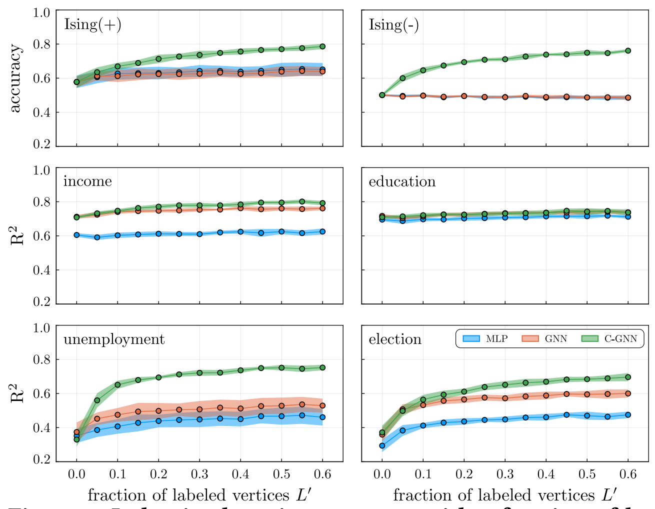

Figure 6: Inductive learning accuracy with a fraction of labeled vertices in the new graph. C-GNN is able to utilize the extra labels more effectively than the baselines, and it does not need neural network fine-tuning.

fair comparison, we then use the learned parameters as the initial guess for G', and optimize the model further with the labels on L'.

Results. We test the performance of our framework and the baselines for different sizes of L' (Fig. 6). C-GNN and GNN gives the same prediction accuracy if no vertex label on G' is available, but, as the number of labeled points increases, C-GNN outperforms the baselines by large margins on multiple datasets. This indicates that the learned residual correlation generalizes well to unseen graphs. Household income and education level predictions do not benefit much from our framework, partially because those statistics are relatively stable over time, so the models trained on 2012 data are already a good proxy for 2016. Remarkably, C-GNN works well without fine-tuning the neural network parameters on the new labels of L', indicating that the feature-label mapping oftentimes shifts from G to G' collectively amongst neighboring vertices.

5 RELATED WORK

By now, semi-supervised learning on graphs has been extensively studied [17, 19, 35, 37]. Label propagation or diffusion "distributes" observed vertex labels throughout the graph [35, 37], but were not designed to incorporate additional vertex features. Laplacian Regularization [1] and Manifold regularization [3] propose to augment feature-based supervised learning methods with an unsupervised loss function that minimize differences between connected vertices. These methods assume neighboring vertices should have similar labels, which is true in many applications.

There are direct models of correlation structure for graph-based semi-supervised learning [33]; such approaches are more computationally expensive and not amenable to joint learning with GNN parameters. The marginalized Gaussian conditional random field (m-GCRF) [27] is closer to our approach, using a CRF to model the label distribution given the vertex features, which reduces to Gaussian distributions with sparse precision matrices under the right choice of potential function. In contrast, we model the correlation of regression residuals instead of the outcomes themselves, and our precision matrix parameterization enables linear-time learning.

The inability of existing GNN approaches to capture label correlations has been discussed in the classification setting. Recent proposals include graph Markov neural networks [23] and conditional graph neural fields [11], which use a CRF to model the joint distribution of vertex classes; as well as positional GNNs [34], which use a heuristic of letting GNN aggregation parameters depend on distances to anchor nodes. With the CRF approaches, the joint likelihood does not have a closed form expression, and such models are trained by maximizing the pseudo-likelihood with the expectation-maximization algorithm. The regression setting here is more mathematically convenient: an unbiased exact joint likelihood estimate can be quickly computed, and the outcome has an interpretable decomposition into base prediction and residual.

6 DISCUSSION

Our semi-supervised regression framework combines the advantages of GNNs and label propagation to get value from both vertex feature information and outcome correlations. Our experiments show that accounting for outcome correlations can give enormous performance gains, especially in cases where the base prediction by a GNN is only mildly accurate. In other words, label correlations can provide information complementary (rather than redundant) to vertex features in some datasets. Understanding this more formally is an interesting avenue for future research.

Our C-GNN uses only a few parameters to model the label correlation structure, and learns the direction and strength of correlations with highly efficient algorithms. The model also enables us to measure uncertainty in predictions, although we did not focus on this. The C-GNN can model more types of data and requires some careful numerical algorithms to scale well. Our simplified LP-GNN approach offers a simple, light-weight add-on to any GNN implementation that can often substantially boost performance.

ACKNOWLEDGEMENTS

This research was supported by NSF award DMS-1830274; ARO award W911NF-19-1-0057; ARO MURI; and JPMorgan Chase & Co.

REFERENCES

[1] Rie Kubota Ando and Tong Zhang. 2006. Learning on Graph with Laplacian Regularization. In *NeurIPS*.
[2] Haim Avron and Sivan Toledo. 2011. Randomized Algorithms for Estimating the Trace of an Implicit Symmetric Positive Semi-Definite Matrix. *J. ACM* (2011).
[3] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. *J. Mach. Learn. Res.* (2006).
[4] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.
[5] Barry A. Cipra. 1987. An Introduction to the Ising Model. *Am. Math. Monthly* (1987).
[6] Kun Dong, Austin R Benson, and David Bindel. 2019. Network density of states. In *KDD*.
[7] David Easley and Jon Kleinberg. 2010. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press.
[8] Juan Fernández-Gracia et al. 2014. Is the Voter Model a Model for Voters? *Physical Review Letters* (2014).
[9] JK Fitzsimons, MA Osborne, SJ Roberts, and JF Fitzsimons. 2018. Improved stochastic trace estimation using mutually unbiased bases. AUAI Press.
[10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Springer.
[11] Hongchang Gao, Jian Pei, and Heng Huang. 2019. Conditional Random Field Enhanced Graph Convolutional Neural Networks. In *KDD*.
[12] Jacob Gardner et al. 2018. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. In *NeurIPS*.
[13] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*.
[14] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin* (2017).
[15] Ken Hayami. 2018. Convergence of the Conjugate Gradient Method on Singular Systems. *NII Technical Reports* (2018).
[16] M.F. Hutchinson. 1989. A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines. *Communications in Statistics - Simulation and Computation* (1989).
[17] Rania Ibrahim and David Gleich. 2019. Nonlinear Diffusion for Community Detection and Semi-Supervised Learning. In *WWW*.
[18] Mike Innes. 2018. Flux: Elegant Machine Learning with Julia. *Journal of Open Source Software* (2018).
[19] Junteng Jia, Michael T. Schaub, Santiago Segarra, and Austin R. Benson. 2019. Graph-Based Semi-Supervised & Active Learning for Edge Flows. In *KDD*.
[20] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[21] Martina Morris and Richard Rothenberg. 2011. *HIV Transmission Network Metastudy Project: An Archive of Data From Eight Network Studies, 1988–2001*. Inter-university Consortium for Political and Social Research.
[22] Mark Newman. 2010. *Networks: An Introduction*. Oxford University Press.
[23] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. GMNN: Graph Markov Neural Networks. *ICML*.
[24] Carl Edward Rasmussen. 2003. Gaussian processes in machine learning. In *Summer School on Machine Learning*. Springer, 63–71.
[25] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. *arXiv preprint arXiv:1909.13021* (2019).
[26] Cosma Shalizi. 2015. Weighted and Generalized Least Squares. https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/24/lecture-24--25.pdf.
[27] Jelena Stojanovic et al. 2015. Semi-supervised learning for structured regression on partially observed attributed graphs. In *ICDM*.
[28] Shashanka Ubaru, Jie Chen, and Yousef Saad. 2017. Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature. *SIAMX* (2017).
[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR* (2018).
[30] David S. Watkins. 2007. *The Matrix Eigenvalue Problem*. Society for Industrial and Applied Mathematics.
[31] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *Nature* (1998).
[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[33] Ya Xu, Justin S Dyer, and Art B Owen. 2010. Empirical Stationary Correlations for Semi-supervised Learning on Graphs: Network Modeling. *AOAS* (2010).
[34] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. *ICML* (2019).
[35] Dengyong Zhou et al. 2004. Learning with Local and Global Consistency. In *NeurIPS*.
[36] Jie Zhou et al. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).
[37] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. 2003. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *ICML*.

A APPENDIX

Here we provide some implementation details of our methods to help readers reproduce and further understand the algorithms and experiments in this paper. All of the algorithms used in this paper are implemented in Julia 1.2. The source code, data, and experiments are available at:

https://github.com/000Justin000/gnn-residual-correlation.

A.1 Label Propagation Algorithm

Given targets on the training vertices $\mathbf{z}_L$, LP computes the targets on the testing vertices $\mathbf{z}_U$ with the following constrained minimization:

$$\mathbf{z}^{\text{LP}} = \arg\min_{\hat{\mathbf{z}}} \hat{\mathbf{z}}^\intercal \mathcal{L} \hat{\mathbf{z}} \qquad \text{s.t.} \qquad \hat{\mathbf{z}}_L = \mathbf{z}_L \tag{23}$$

where $\mathcal{L} = \mathbf{I} - \mathbf{S}$ is the normalized Laplacian matrix. This is the method by Zhu et al. [37] but with the normalized Laplacian instead of the combinatorial Laplacian, which is nearly the same as the approach by Zhou et al. [35], except targets on L are fixed. The solution on the unlabeled vertices is

$$\mathbf{z}_U^{\text{LP}} = -\mathcal{L}_{UU}^{-1} \mathcal{L}_{UL} \mathbf{z}_L, \tag{24}$$

which we can compute with CG. If L and U are disconnected, $\mathcal{L}_{UU}$ is singular. Then starting with an all-zero initial guess, CG converges to the minimal norm solution that satisfies Eq. (23) [15]. The entire algorithm is summarized in Algorithm 4.

Algorithm 4: Label Propagation.

```
Input  : normalized adjacency matrix S; training targets $\mathbf{z}_L$
         (label or residual);
Output : predicted targets $\mathbf{z}_U^{\text{LP}}$ for unknown vertices
1 $\mathcal{L} \leftarrow \mathbf{I} - \mathbf{S}$                                ▷ precision matrix
2 $\mathbf{z}_U^0 \leftarrow \mathbf{0}$                                  ▷ initial guess
3 $\mathbf{z}_U^{\text{LP}} \leftarrow \texttt{ConjugateGradient}(\mathcal{L}_{UU}, -\mathcal{L}_{UL}\mathbf{z}_L, \mathbf{z}_U^0)$
```

A.2 Stochastic logdet Estimation with `Flux.jl`

The base GNN regressors are implemented in Julia with `Flux.jl` [18]. For better compatibility with the underlying GNN, we implement the stochastic estimation algorithm using the "customized gradient" interface provided by `Flux.jl`. For example, Listing 1 shows the code snippet that defines the log-determinant computation: when `logdetΓ` is invoked, its output is tracked and its derivative can be computed automatically with back-propagation. This scheme greatly simplifies the downstream implementations for data mining experiments, and it works as if we were computing the exact gradient — only orders of magnitude faster with a minor loss of accuracy, as evidenced by our experiments in Section 3.2.

A.3 Additional Details on Experimental Setup

Neural network architecture. Our regression pipeline first encodes each vertex into an 8-dimension representation using an MLP or GNN and then uses a linear output layer to predict its label. For the MLP, we use a 2-hidden-layer feedforward network with 16 hidden units and ReLU activation function. Each GNN we consider also consists of 2 layers, each with 16 hidden units and ReLU activation function.

```julia
using Flux.Tracker: track, @grad

# When this function is invoked, Flux automatically
# tracks the output for auto-differentiation
logdetΓ(α, β; S, t, k) = track(logdetΓ, α, β; S=S, t=t, k=k);

# This tells Flux how to track logdetΓ
@grad function logdetΓ(α, β; S, t, k)
    """
    Input:
      α: (vector of) model parameters
      β: model parameter
      S: (vector of) normalized adjacency matrices
      t: # of trial vectors
      k: # of Lanczos tridiagonal iterations

    Output:
      1): logdet(Γ)
      2): map from sensitivity of logdet(Γ)
               to sensitivity of α, β
    """
    # sample Gaussian random vector
    n = size(S,1);
    Z = randn(n,t);

    # eqns (5) in this paper
    Γ = getΓ(α, β; S=S);
    ∂Γ∂α = get∂Γ∂α(α, β; S=S);
    ∂Γ∂β = get∂Γ∂β(α, β; S=S);

    # adopted from Gardner 2018 GPytorch paper
    X, TT = mBCG(Y->Γ[P,P]*Y, Z; k=k);

    # eqn (20) in this paper
    vv = 0;
    for T in TT
        eigvals, eigvecs = eigen(T);
        vv += sum(eigvecs[1,:].^2 .* log.(eigvals));
    end
    Ω = vv*n/t;

    # eqn (18) in this paper
    trΓiM(M) = sum(X.*(M[P,P]*Z))/t;
    ∂Ω∂α = map(trΓiM, ∂Γ∂α);
    ∂Ω∂β = trΓiM(∂Γ∂β);

    return Ω, Δ -> (Δ*∂Ω∂α, Δ*∂Ω∂β);
end
```

Listing 1: Code snippet for estimating the log-determinant and its derivatives.

Optimization. For all but the Twitch-PT datasets, the framework parameters α, β are optimized using gradient descent with learning rate 10^{-1}. The Twitch-PT dataset uses a limited-memory BFGS optimizer. For the MLP and GNN experiments summarized in Table 1 and Fig. 5, the neural network parameters are optimized for 75 epochs using the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and learning rate 10^{-3}. For the inductive experiments summarized in Fig. 6, the neural network parameters are further fine-tuned for 25 epochs with the Adam optimizer at a smaller learning rate 5×10^{-4}. All of our experiments are performed on a single workstation with an 8-core i7-7700 CPU @ 3.60GHz processor and 32 GB memory.

A.4 Additional Details on Datasets

Ising model simulations. The Ising model samples random labels on a two-dimensional grid graph. For each vertex i, there is a discrete variable $\sigma_i \in \{-1, +1\}$ representing its spin state. A spin configuration σ assigns a spin state to every vertex in the graph. The Ising model considers two type of interactions: (i) interaction

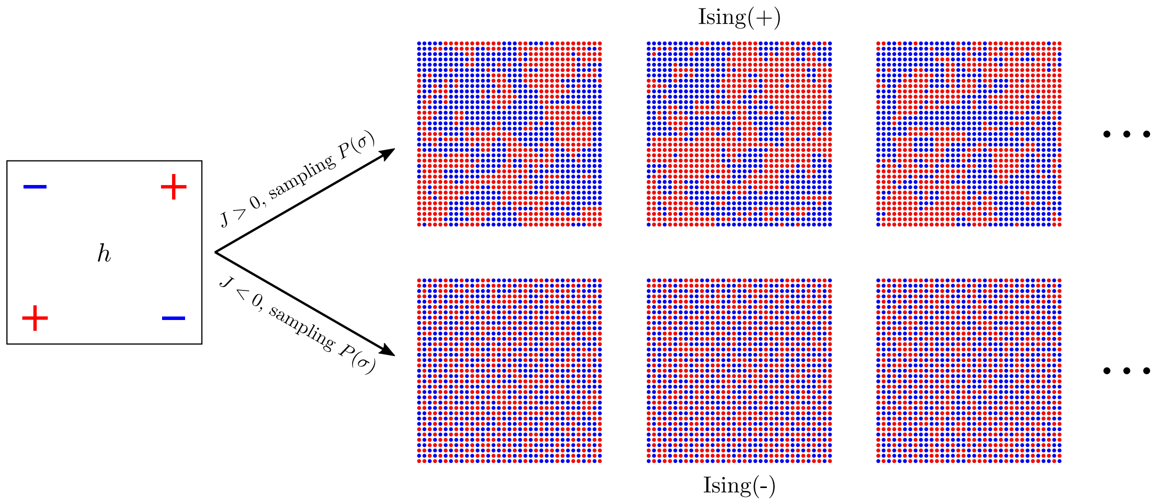

Figure 7: Example Ising spin configurations sampled from the Boltzmann distribution. Vertices with $+1$ spins are colored in red, and vertices with -1 spin are colored in blue.

Table 2: Transductive learning accuracy of our framework, using different GNN architecture as the base predictor.

	GCN [20]			GraphSAGE-max [13]			GraphSAGE-pooling [13]		
Dataset	GNN	LP-GNN	C-GNN	GNN	LP-GNN	C-GNN	GNN	LP-GNN	C-GNN
Ising(+)	0.61 ± 0.04	0.72 ± 0.03	0.72 ± 0.03	0.67 ± 0.04	0.76 ± 0.02	0.76 ± 0.02	0.67 ± 0.04	0.76 ± 0.02	0.76 ± 0.02
Ising(-)	0.47 ± 0.02	0.34 ± 0.02	0.70 ± 0.03	0.47 ± 0.02	0.30 ± 0.03	0.77 ± 0.03	0.48 ± 0.02	0.30 ± 0.03	0.77 ± 0.02
income	0.60 ± 0.04	0.61 ± 0.05	0.62 ± 0.04	0.73 ± 0.03	0.79 ± 0.03	0.78 ± 0.04	0.74 ± 0.03	0.78 ± 0.03	0.77 ± 0.02
education	0.45 ± 0.04	0.44 ± 0.04	0.47 ± 0.04	0.67 ± 0.02	0.70 ± 0.02	0.70 ± 0.03	0.68 ± 0.02	0.70 ± 0.03	0.70 ± 0.03
unemployment	0.49 ± 0.03	0.72 ± 0.03	0.72 ± 0.03	0.57 ± 0.05	0.74 ± 0.04	0.75 ± 0.05	0.60 ± 0.05	0.74 ± 0.04	0.75 ± 0.04
election	0.45 ± 0.03	0.61 ± 0.02	0.60 ± 0.02	0.43 ± 0.04	0.64 ± 0.03	0.65 ± 0.03	0.49 ± 0.06	0.66 ± 0.02	0.65 ± 0.02
Anaheim	0.69 ± 0.05	0.75 ± 0.05	0.75 ± 0.05	0.73 ± 0.04	0.79 ± 0.05	0.80 ± 0.04	0.74 ± 0.04	0.80 ± 0.04	0.80 ± 0.05
Chicago	0.58 ± 0.05	0.63 ± 0.05	0.63 ± 0.05	0.64 ± 0.05	0.68 ± 0.05	0.68 ± 0.05	0.66 ± 0.05	0.69 ± 0.04	0.68 ± 0.04
sexual	0.77 ± 0.04	0.72 ± 0.04	0.92 ± 0.02	0.86 ± 0.02	0.86 ± 0.02	0.92 ± 0.02	0.85 ± 0.05	0.85 ± 0.04	0.92 ± 0.02
Twitch-PT	0.54 ± 0.02	0.65 ± 0.01	0.64 ± 0.02	0.69 ± 0.04	0.69 ± 0.04	0.70 ± 0.03	0.72 ± 0.03	0.72 ± 0.03	0.71 ± 0.03

between the spin of each vertex with external field and (ii) the interaction between neighboring spins. Those interactions constitute the "energy" for each spin configuration:

$$H(\sigma) = -\sum_{(i,j)\in E} J_{ij}\sigma_i\sigma_j - \sum_{i\in V} h_i\sigma_i, \tag{25}$$

where J_{ij} controls the interaction between neighboring vertices, and h_i denotes the external field on vertex i. Finally, the configuration probability is given by the Boltzmann distribution,

$$P(\sigma) = \frac{e^{-H(\sigma)}}{\sum_{\sigma'} e^{-H(\sigma')}}. \tag{26}$$

Our Ising spin simulation randomly draws from this Boltzmann distribution. For the Ising(+) dataset, we set $J_{ij} = J = 0.1$ and $h_i = 0.35 \cdot (x_i)_1 \cdot (x_i)_2$, where $\mathbf{x}_i$ is the coordinate of vertex i normalized between -1.0 and $+1.0$. In other words, the system favors parallel spins between neighboring vertices, and the external field exhibits an "XNOR" spatial pattern. For the Ising(-) dataset, a similar setting is used, except that $J_{ij} = J = -0.1$. Some sampled Ising spin configurations are shown in Fig. 7.

Sexual interaction dataset. The dataset used to construct the sexual interaction network was collected by Colorado Springs project 90, which details the relationships of 7,674 individuals. We take the largest connected component in the derived sexual relation network, which consists of 1,888 vertices and 2,096 edges. Of the 2,096 relationships, 2,007 are heterosexual and 89 are homosexual.

A.5 Other GNN Base Predictors

We tested a variety of GNN architectures as base regressors in our framework, and Table 2 summarizes the results. Here, we see the exact same trend as describe in Section 4.2: C-GNN substantially out-performs the base GNN on almost all datasets, and LP-GNN outperforms GNN on datasets where vertex labels are positively correlated. These experiments support our claim that the performance gains we observe from exploiting label correlation is robust to change of the underlying GNN architecture.