Learning Latent Structures in Network Games via Data-Dependent Gated-Prior Graph Variational Autoencoders

Xue Yu [1 2] **Muchen Li** [3] **Yan Leng** [4] **Renjie Liao** [5 6 7]

Abstract

In network games, individuals interact strategically within network environments to maximize their utilities. However, obtaining network structures is challenging. In this work, we propose an unsupervised learning model, called data-dependent gated-prior graph variational autoencoder (GPGVAE), that infers the underlying latent interaction type (strategic complement vs. substitute) among individuals and the latent network structure based on their observed actions. Specially, we propose a spectral graph neural network (GNN) based encoder to predict the interaction type and a data-dependent gated prior that models network structures conditioned on the interaction type. We further propose a Transformer based mixture of Bernoulli encoder of network structures and a GNN based decoder of game actions. We systematically study the Monte Carlo gradient estimation methods and effectively train our model in a stage-wise fashion. Extensive experiments across various synthetic and real-world network games demonstrate that our model achieves state-of-the-art performances in inferring network structures and well captures interaction types.

1. Introduction

Social influence and strategic dependencies are prevalent in systems like social networks (Aral et al., 2009; Banerjee et al., 2013; Leng et al., 2023b). These social dynamics can be encapsulated in what are known as *network games*. In these games, an individual's utility is shaped not only by their actions but also by the actions of others in their social circles. Two types of strategic relationships, known as *strategic complement* and *strategic substitute*, encompass a wide range of behaviors. In scenarios involving strategic complements, an individual's actions positively influence others' utility, hence typically leading to higher action levels. This interdependence often results in individuals mimicking the actions of their neighbors. Conversely, in strategic substitute relationships, neighbors' action levels reduce individuals' utility and hence reduce their action levels. In both strategies, the network structure plays a fundamental role in determining individuals' utility and subsequently their decisions. However, it is increasingly common that despite having abundant data on individual decisions, the intricate relationships (*e.g.*, represented by an interaction network) among them often remain concealed due to the high costs of observation or privacy considerations (Leng et al., 2020; Rossi et al., 2022; Leng et al., 2023a). As such, revealing latent network structures and identifying the types of interactions are crucial in network games.

Many studies rely on utility functions to analyze individuals' Nash equilibrium behaviors and estimate the latent network structures (Honorio & Ortiz, 2015; Barik & Honorio, 2019; Leng et al., 2020). These works often assume specific forms of utility functions. Meanwhile, supervised (Belilovsky et al., 2016; Zhao et al., 2023) and semi-supervised approaches (Elinas et al., 2020; Liu et al., 2023; Wang et al., 2023) have been proposed to predict network structures given users' actions and partial network connections (required in semi-supervised learning). In real-world applications, determining the exact form of utility functions or acquiring graph structures is often challenging, which motivates us to explore unsupervised methods that can infer network structures without knowing the utility function.

In this paper, we propose an unsupervised learning model, dubbed *data-dependent Gated-Prior Graph Variational Autoencoder* (GPGVAE), which directly infers latent network structures from game actions. Specifically, in our efforts to capture interaction types such as strategic complements and substitutes, we initially investigate data-dependent priors,

[1]Center for Applied Statistics, School of Statistics, Renmin University of China, Beijing, China [2]Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing [3]Department of Computer Science, University of British Columbia, Vancouver, BC, Canada [4]McCombs School of Business, The University of Texas at Austin, Austin, TX, USA [5]Vector Institute for AI [6]Canada CIFAR AI Chair [7]Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada. Correspondence to: Xue Yu <xueyu_2019@ruc.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

including correlations and anticorrelations, derived from game actions within the framework of Graph Variational Autoencoders (Graph VAEs). We observe that correlation priors help infer latent network structures in complementary interactions, while anticorrelation priors are beneficial for substitute interactions. Given that no single prior consistently dominates across all game settings, we are motivated to introduce a latent interaction type and design a gated (mixture) prior, which alternates between these two priors conditioned on the interaction type. Furthermore, our analysis of the covariance matrix of game actions reveals that games with strategic complement and substitute interactions have distinctive spectral patterns. This finding has led us to design a spectral GNN-based encoder to effectively infer the latent interaction type. We have also developed a Transformer-based graph encoder to predict the latent network structure, an MLP-based encoder to predict user embeddings, and a GNN-based decoder for reconstructing game actions. Lastly, given that our model incorporates both continuous and discrete latent variables, we employ a stage-wise training strategy, *i.e.*, 1) pretraining of the interaction type encoder on synthetic network games and 2) training the rest of the model. Owing to the model design, our training objectives are invariant to permutations of users. We also systematically investigate the Monte Carlo gradient estimation methods for training our model. Extensive experiments on synthetic and real-world datasets demonstrate that our approach achieves state-of-the-art performances.

2. Related Work

The challenge of inferring network structures has attracted extensive attention across different domains. In statistics, the inference of network structures can be framed as an estimation problem for sparse undirected graph models, where the most renowned method is the graphical lasso (Meinshausen & Bühlmann, 2006). However, as highlighted by Pu et al. (2021), selecting regularization parameters in the graphical lasso is challenging. To address this, they propose to replace the regularization term with learnable VAEs and improve its effectiveness and robustness. Shrivastava et al. (2020) introduce a neural network that dynamically adjusts the regularization parameters and other hyperparameters. An unsupervised version is provided in their follow-up work (Shrivastava et al., 2022).

In game theory, research often relies on established forms of utility functions or predefined network structures (Honorio & Ortiz, 2015; Ghoshal & Honorio, 2016). Bramoullé et al. (2014) study Nash equilibrium states and the impact of individual decisions on others' payoffs using two linear utility functions. Barik & Honorio (2019) introduce a block-norm regularization method that can reconstruct Nash equilibrium states and underlying network structures of graphical games. Leng et al. (2020; 2023a) focus on quadratic utility functions and propose an optimization algorithm for inferring network structures and individual marginal benefits based on continuous equilibrium behaviors. Relying on a subset of network structures for supervised learning, Rossi et al. (2022) introduce a Transformer-based method to infer network structures from individual actions.

In machine learning, Belilovsky et al. (2016) propose convolutional neural networks (CNNs) to learn a mapping from sample covariance matrices to graphs in a supervised fashion. However, models learned through this approach lack permutation equivariance. Yu et al. (2019) employ VAEs for inferring Directed Acyclic Graphs (DAGs), which limits its applicability to networks with cycles. Some supervised and semi-supervised methods combine graph learning with downstream tasks and need node labels or partial network connections to learn structures (Franceschi et al., 2019; Elinas et al., 2020; Wu et al., 2022; Zhang et al., 2023; Zou et al., 2023). Kipf et al. (2018) propose an unsupervised method that uses VAEs to infer interactions between individuals and predict the motion trajectories of nodes in a dynamic system. Xu et al. (2023) construct a differentiable k-nearest-neighbor (k-NN) graph on selected features by minimizing the Dirichlet energy, where the parameter k needs to be predefined.

3. Strategic Interactions and Network Games

We now discuss and analyze two primary types of strategic interactions, *i.e.*, strategic complement and strategic substitute, which cover a broad spectrum of real-world behaviors, ranging from consumer consumption behaviors, and investment decisions (Ballester et al., 2006; Bramoullé et al., 2014; Galeotti et al., 2020). We then introduce several popular network games, which provide a mathematical framework to analyze multi-agent behaviors. Within this framework, rational agents make their decisions to maximize their utility, which is affected by their network connections.

Notations. Consider an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of N individuals and $\mathcal{E}$ denotes the relationship set. This network can be represented by a real and symmetric adjacency matrix $A \in \mathbb{R}^{N \times N}$. For any pair of individuals (i, j), $A_{ij} = A_{ji} = 1$ if $(i, j) \in \mathcal{E}$, and otherwise 0. The normalized adjacency matrix is defined as $\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where D is a degree matrix with $D_{ii} = \sum_j A_{ij}$. The normalized graph Laplacian matrix is defined by $L = I - \tilde{A}$. u_i is the utility function of individual i and $\mathcal{N}_i$ is the set of its neighbors. x_i denotes the action taken by individual i in a game. In the real world, an action can be continuous (*e.g.*, the amount of time to spend on a social platform) or discrete (*e.g.*, adopting a product). In this paper, we consider continuous actions. For any games, we assume that the action of individual i is drawn from

the same model in an i.i.d fashion. $x^* = [x_1^*, ..., x_N^*]^\top$ denotes the equilibrium action of all individuals when the Nash equilibrium condition is satisfied.

3.1. Strategic Complement

In strategic complementary games, if individuals i and j are connected, their actions are expected to be positively correlated. We now introduce three widely-used network games that exhibit strategic complements.

BH graphical games. Barik & Honorio (2019) develop a graphical game in which individuals' utilities are affected by the average actions of their neighbors. They consider a utility function in the following form:

$$u_i = -\|x_i - \sum\nolimits_{j\in\mathcal{V}} \tilde{A}_{ij} x_j\|_2. \tag{1}$$

To maximize their utility, individuals are incentivized to conform to the actions of their neighbors, because any deviations from their neighbors' actions can reduce their utility u_i. In this type of game, the interactions are purely strategic complements and the equilibrium action x^* is the largest eigenvector corresponding to the largest eigenvalue of $\tilde{A}$.

Linear Influence Games. Based on the linear influence games with discrete actions (Irfan & Ortiz, 2011), Rossi et al. (2022) consider the variant with continuous behaviors. Linear influence games can represent complex contagion behaviors (Centola & Macy, 2007). In this game, individuals' adoption decisions are affected by their actions and those of their neighbors, as well as an idiosyncratic "threshold" b_i. If the aggregated impact from i's neighbors is lower than b_i, then the behavior of i does not change. The continuous version of the utility function can be represented as

$$u_i = \sum\nolimits_{j\in\mathcal{N}_i} \tilde{A}_{ij} x_i x_j - b_i x_i. \tag{2}$$

Considering the homophily effect in the marginal benefit b, Rossi et al. (2022) investigate its covariance, determined by the network structure, affects the distribution of actions. Here, b follows a Gaussian distribution:

$$b \sim \mathcal{N}(0, L_\alpha^\dagger), \quad L_\alpha = (1-\alpha)I + \alpha L = I - \alpha\tilde{A}. \tag{3}$$

where $b = [b_1, ..., b_N]^\top \in \mathbb{R}^N$, $I \in \mathbb{R}^{N\times N}$ is the identity matrix, $\dagger$ represents the pseudo-inverse and $\alpha \in [0, 1]$. When $\alpha = 1$, individuals tend to connect with others who have similar inherent characteristics; When $\alpha = 0$, each individual's idiosyncratic b_i is independent of its neighbors. Given the distribution (3), the equilibrium action x^* obeys a Gaussian distribution, *i.e.*, $x^* \sim \mathcal{N}(0, \tilde{A}^{-1}(I-\alpha\tilde{A})^\dagger\tilde{A}^{-1})$, where $\tilde{A}$ is assumed to be invertible. Interactions in this game are usually strategic complements.

Linear Quadratic Games. Ballester et al. (2006); Bramoullé et al. (2014) propose linear quadratic games where an individual i chooses its action x_i by maximising the following quadratic payoff u_i:

$$u_i = b_i x_i - \frac{1}{2}x_i^2 + \beta \sum_{j\in\mathcal{N}_i} \tilde{A}_{ij} x_i x_j. \tag{4}$$

The third term $\beta\sum_{j\in\mathcal{N}_i}\tilde{A}_{ij}x_ix_j$ represents the impact of the actions of other neighbors on individual i, where β controls the strength of influence from its neighbors. To ensure the uniqueness and stability of the equilibrium behavior, it is commonly assumed that the spectral radius $\rho(\beta\tilde{A}) < 1$ (*i.e.*, $|\beta| < 1$). Additionally, when $\beta > 0$, the action of one individual $j \in \mathcal{N}_i$ positively affects the payoff of individual i when taking the same actions, and individual i tends to take an action consistent with that of its neighbors. This is a strategic complement. For this game, the equilibrium action x^* follows

$$x^* \sim \mathcal{N}(0, (I-\beta\tilde{A})^{-1}(I-\alpha\tilde{A})^\dagger(I-\beta\tilde{A})^{-1}). \tag{5}$$

3.2. Strategic Substitute

In addition to strategic complement, individuals' decisions can be strategic substitutive of each other (Bulow et al., 1985; Goyal & Moraga-González, 2001). In strategic substitute games, a heightened action level by individual i reduces the payoff for individual j. Hence, actions are expected to be negatively correlated in these games when i and j are connected. A canonical example of strategic substitute games is a linear quadratic game with $\beta < 0$. Specifically, in Eq. (4), if $\beta < 0$, a higher level of action by individual $j \in \mathcal{N}_i$ negatively affects the payoff of individual i when taking a higher level of actions, hence individuals have the incentive to reduce their action level. We provide more details about the three types of games in Appendix A.1.

4. Data-Dependent Gated-Prior Graph VAE

In the aforementioned network games, the network structure directly influences the distribution of individuals' equilibrium actions. This suggests that one could infer the underlying unobservable network structure from individuals' equilibrium actions. To accomplish this goal, we propose a generative model with network structures and the interaction type being latent variables, dubbed GPGVAE. In the following, we will first introduce the overall framework. Then we will describe the designs of data-dependent gated prior, the encoder, and the decoder respectively.

4.1. Graph VAE Framework

Given a set of N individuals/users, we sample their actions $X \in \mathbb{R}^{N\times M}$ from M games with the same latent network structure, *i.e.*, the binary adjacency matrix A. We assume that the observed actions X satisfy the Nash equilibrium condition. Although this assumption may be violated in some practical cases, we will see that our framework still performs empirically well. To achieve the goal of inferring

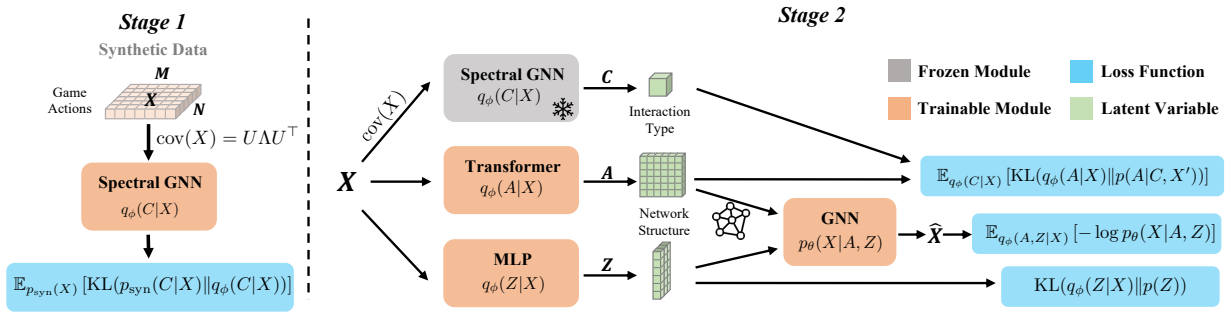

Figure 1. The overview of our data-dependent Gated Prior Graph VAE. We employ a 2-stage training strategy: 1) pretrain our spectral GNN-based interaction type encoder $q_\phi(C|X)$; 2) train the model with $q_\phi(C|X)$ being fixed.

the latent network structure from user actions, we develop a latent graph VAE framework which consists of three encoders, a decoder, and a prior. Furthermore, to account for the two types of interactions (complement vs. substitute), we introduce a binary latent variable $C \in \{0, 1\}$. We additionally introduce a latent vector per user for capturing other potential factors, denoted as $Z \in \mathbb{R}^{N\times D}$ where D is the hidden dimension. To ensure the tractability of the encoder $q_\phi(A, Z, C|X)$, we assume the conditional independence $q_\phi(A, Z, C|X) = q_\phi(A|X)q_\phi(Z|X)q_\phi(C|X)$, where the learnable parameters are absorbed in ϕ. The decoder is denoted as $p_\theta(X|A, Z, C)$ with learnable parameters θ. We propose the data-dependent gated prior as $p(A, Z, C) := p(A|C, X')p(C)p(Z)$. The prior of the network structure A is data-dependent since it is conditioned on a subset of observed data X'. Moreover, the binary interaction type C serves as a gating variable that alternates between two mixture distributions of the network structure. Similar to other VAEs, the learning is to minimize the negative evidence lower bound (ELBO):

$$\begin{aligned}\mathcal{L}(\phi,\theta) = &\ \mathbb{E}_{q_\phi(A,Z,C|X)}\left[-\log p_\theta(X|A,Z,C)\right] \\ &+ \mathrm{KL}(q_\phi(Z|X)\|p(Z)) + \mathrm{KL}(q_\phi(C|X)\|p(C)) \\ &+ \mathbb{E}_{q_\phi(C|X)}\left[\mathrm{KL}(q_\phi(A|X)\|p(A|C,X'))\right], \qquad (6)\end{aligned}$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence. The first term of the right-hand side is the so-called reconstruction loss. We will introduce the motivation and the design of these components in the subsequent sections.

4.2. Data-Dependent Gated Prior of Network Structures

We now introduce our data-dependent gated prior. Let us first consider a simpler prior design $p(A, Z) = p(A)p(Z)$, which will motivate the necessity of a data-dependent prior and the introduction of the latent interaction type C. $p(Z)$ could be simply set as standard Normal. The encoder and decoder in this case are simplified as $q_\phi(A, Z|X) = q_\phi(A|X)q_\phi(Z|X)$ and $p_\theta(X|A, Z)$ respectively. We then compare different choices of $p(A)$ via synthetic experiments with 50 Erdős-Rényi (ER) graphs (each with 20 nodes and

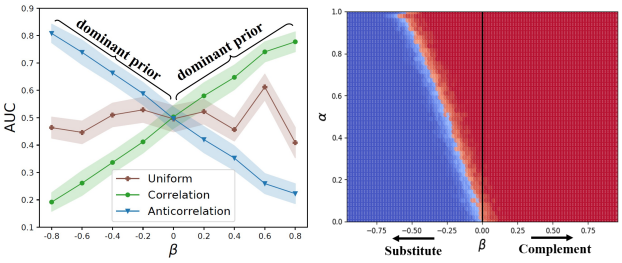

Figure 2. Dominant priors (*i.e.*, a prior leads to the highest AUC) on linear quadratic games. Left: AUCs of different priors under varying β and $\alpha = 0$. Right: Dominant priors (red: correlation; blue: anticorrelation) under varying α and β.

100 games) and actions X drawn from linear quadratic games. In particular, we assume edge-independent prior $p(A) = \prod_{i,j} p(A_{ij})$ where $p(A_{ij})$ takes the following forms: 1) *uniform*: $p(A_{ij} = 1) = 1/2$; 2) *correlation*: $p(A_{ij} = 1) = \psi_{\mathrm{corr}}(X_i, X_j) := (\rho_{ij} + 1)/2$; 3) *anticorrelation*: $p(A_{ij} = 1) = 1 - \psi_{\mathrm{corr}}(X_i, X_j) = (1 - \rho_{ij})/2$. Here we use Pearson's correlation coefficient between users i and j based on their sampled actions,

$$\rho_{ij} = \frac{\sum_{k=1}^{M}(X_{ik} - \bar{X}_i)(X_{jk} - \bar{X}_j)}{\sqrt{\sum_{k=1}^{M}(X_{ik} - \bar{X}_i)^2}\sqrt{\sum_{k=1}^{M}(X_{jk} - \bar{X}_j)^2}},$$

where $\bar{X}_i = (\sum_{k=1}^{M} X_{ik})/M$. We vary the prior choice with the rest of the model unchanged and then train these models. More details are provided in Appendix D.1.

The left subplot of Figure 2 presents the averaged area under the curve (AUC) values and the standard deviation (std) for inferred latent network structures compared to the ground truth. We fix the marginal benefit parameter $\alpha = 0$ and vary the strength of dependencies between individuals β from -0.8 to 0.8. It is clear that the model based on a uniform prior yields an AUC value around 0.5, whereas correlation and anticorrelation priors dominate in certain regions of β. Here, a "dominant prior" means the one that leads to a higher AUC. More details can be found in Appendix A.2. The right subplot of Figure 2 shows the dominant priors on

linear quadratic games with varying α and β. The correlation prior (red) consistently dominates under the strategic complement ($\beta > 0$) cases, whereas the anticorrelation (blue) dominates in the majority of strategic substitute ($\beta < 0$) cases. In other words, a correlation prior effectively recovers the underlying network structure in games with strategic complement, and an anticorrelation prior does so in cases of strategic substitute. Notably, the red inverted triangle area shows that there exist games that the correlation prior still dominates under strategic substitute interactions. For linear influence and BH graphical games, which are of strategic complements, the dominant prior is mainly the correlation prior.

The above experiments highlight a few key insights. 1) Data-dependent priors (*e.g.*, correlation and anticorrelation) are preferable to commonly used uninformative priors (*e.g.*, uniform); 2) The correlation and anticorrelation priors are good surrogates for indicating strategic complement and substitute interactions respectively; 3) Knowing the interaction type helps us identify the dominant prior.

Therefore, it is natural to design a more flexible prior, *e.g.*, a mixture prior, that consistently dominates under different game settings. We estimate the correlation and anticorrelation from a subset of data X' and introduce a binary latent variable C to switch between them. Specifically, we have.

$$p(A_{ij}|C=1, X') = \text{Ber}(A_{ij}|\psi_{\text{corr}}(X'_i, X'_j)), \quad (7)$$

where $\text{Ber}(\cdot|a)$ is a Bernoulli distribution with parameter a. The correlation prior naturally captures the effect of positive interactions, *i.e.*, the more similar the actions among two users, the more likely they are connected. Conversely, we can define an anticorrelation prior:

$$p(A_{ij}|C=0, X') = \text{Ber}(A_{ij}|1-\psi_{\text{corr}}(X'_i, X'_j)). \quad (8)$$

The probability of an edge existing between two individuals increases when they exhibit distinct behaviors.

However, it is non-trivial to set $p(C)$ since a uniform prior tends to achieve the average AUC between correlation and anticorrelation. We bypass this issue via pretraining an encoder $q_\phi(C|X)$ to predict the interaction type C. By doing so, we then fix this encoder during the training of the model. In other words, the term $\text{KL}(q_\phi(C|X)\|p(C))$ in Eq. (6) is constant, thus avoiding choosing the right $p(C)$. We will discuss more details in Section 5.

4.3. Spectral GNN based Encoder

We now introduce our encoder. As aforementioned, it is factorized as $q_\phi(A, Z, C|X) = q_\phi(A|X)q_\phi(Z|X)q_\phi(C|X)$, where latent variables Z, A, and C are the user embedding, the network structure, and the interaction type respectively.

User Embedding Encoder. For the user embedding Z, we design its posterior as conditionally independent Normal,

$$q_\phi(Z|X) = \prod_{i=1}^{N} q_\phi(Z_i|X_i) = \prod_{i=1}^{N} \mathcal{N}(Z_i; \mu_i, \text{diag}(\sigma_i^2)) \quad (9)$$

where μ and $\log \sigma^2$ are output by MLPs that take X as input. Due to the conditional independence and the permutation equivariance of MLPs, $q_\phi(Z|X)$ is invariant to permutations of users. The details are included in Appendix B.

Network Structure Encoder. For the network structure A, we use a mixture of Bernoulli distributions:

$$q_\phi(A|X) = \sum_{k=1}^{K} \alpha_k \prod_{i \le j} \theta_{k,ij}^{A_{ij}} (1-\theta_{k,ij})^{(1-A_{ij})}, \quad (10)$$

where K is the number of mixture components. $\theta_{k,ij}$ is the parameter of edge (i, j) of the k-th component. α_k represents the probability of selecting the k-th component. When $K = 1$, the distribution simplifies to a Bernoulli distribution where edges are independent given the actions. When $K > 1$, the dependence of edges can be effectively modeled. We build a Transformer (Vaswani et al., 2017) based encoder that predicts α_k and $\theta_{k,ij}$ using the input X. Since we do not use positional encoding and the rest of Transformer is permutation-equivariant, this encoder is invariant to permutations of users. We leave the detailed architecture design in Appendix B.

Spectral Interaction Encoder. As seen before, choosing the correct interaction type C is essential to achieve the dominant prior. However, as we will see later that straightforward designs (*e.g.*, MLPs) of the encoder $q_\phi(C|X)$ perform poorly under certain game settings. To motivate our solution, let us consider linear quadratic games. Define the covariance matrix of actions X as $\text{cov}(X) \in \mathbb{R}^{N\times N}$, and $\text{cov}(X)_{ij} = \sum_{k=1}^{M}(X_{ik} - \bar{X}_i)(X_{jk} - \bar{X}_j)$. The eigendecomposition of $\text{cov}(X) = U\Lambda U^\top$, where $\Lambda = \text{diag}(\lambda_1, ..., \lambda_N)$. We then plot $\log(\lambda_{\max}/\sum_{i=1}^{N}\lambda_i)$ under different parameter settings in the left subplot of Figure 3. As the positive interactions among individuals become stronger, *i.e.*, $\beta \to 1$, $\alpha \to 1$, the largest eigenvalue dominates other eigenvalues more significantly. This would result in an individual's action and those of its neighbors becoming closer, converging to the leading eigenvector associated with the largest eigenvalue. We can also view this from the perspective of graph signal processing (Dong et al., 2020). Based on the eigendecomposition $\tilde{A} = \tilde{U}\tilde{\Lambda}\tilde{U}^\top$, we can rewrite the covariance as,

$$\text{cov}(X) = (I-\beta\tilde{A})^{-1}(I-\alpha\tilde{A})^{\dagger}(I-\beta\tilde{A})^{-1} = \tilde{U}f(\tilde{\Lambda})\tilde{U}^\top,$$

where $f(\tilde{\Lambda}) = \text{diag}(f(\tilde{\lambda}_1), ..., f(\tilde{\lambda}_N))$ and the spectral filter is $f(\tilde{\lambda}_i) = 1/(1-\beta\tilde{\lambda}_i)^2(1-\alpha\tilde{\lambda}_i)$. Since $\tilde{\lambda}_i \in [-1, 1]$, when $\beta \to 1$ and $\alpha \to 1$, $f(\cdot)$ assigns higher weights to larger eigenvalues, *i.e.*, $\tilde{\lambda}_i \to 1$. This is essentially performing low-pass filtering since large eigenvalues correspond to low frequency. Taking the limit, this implies the same result

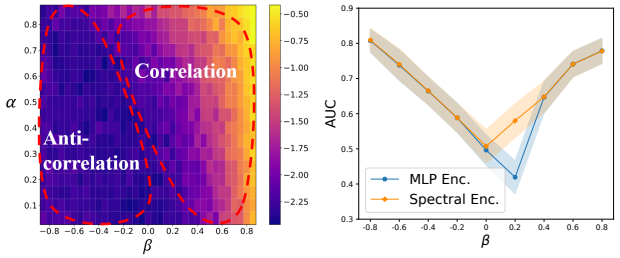

Figure 3. Left: Contribution of the largest eigenvalue $\lambda_{\max}$ of $\text{cov}(X)$ under different settings. The colour bar represents the logarithmic value $\log(\lambda_{\max}/\sum_{i=1}^{N}\lambda_i)$. Right: The AUC results on linear quadratic games obtained with two encoders.

as before, *i.e.*, individuals' actions converge to the leading eigenvector. When $\beta \to -1$ and $\alpha \to 0$, it performs high-pass filtering and the actions of individuals are far away from their neighbors.

The two annotated areas in the left subplot of Figure 3 indicate regions dominated by correlation and anticorrelation priors. Notably, there is a similarity between the patterns of dominant priors in Figure 2 and those of the largest eigenvalue contributions in Figure 3. This inspires us to design an interaction encoder $q_\phi(C|X)$ based on the spectral pattern of the covariance $\text{cov}(X)$. Specifically, following the state-of-the-art spectral GNNs in (Bo et al., 2023; Lim et al., 2023), we employ the range encoding for eigenvalues and set-to-set spectral filters. We compare the two encoder designs, *i.e.*, MLPs vs. Spectral GNNs, in the right subplot of Figure 3. Moreover, we show that the encoder is invariant to permutations of users as the spectral GNNs are equivariant to permutations. More details are provided in Appendix B. One can see that these two encoders perform similarly well in settings where strategic complement and substitute patterns are easy to be identified, *i.e.*, $\beta \leq 0$ or $\beta \geq 0.4$. For the region where the two patterns are hard to distinguish, *i.e.*, $0 < \beta < 0.4$, our spectral GNNs based encoder significantly outperforms the MLP based one.

4.4. Decoder

To reconstruct actions, we need a decoder $p_\theta(X|A, Z, C)$. Since the network structure A already depends on the interaction type C in our data-dependent gated prior, we assume $p_\theta(X|A, Z, C) = p_\theta(X|A, Z)$ for simplicity. We also assume conditional independence among different users, *i.e.*, $p_\theta(X|A, Z) = \prod_{i=1}^{N} p_\theta(X_i|A, Z_i) = \prod_{i=1}^{N} \mathcal{N}(X_i; \mu_i, \text{diag}(\sigma_i^2))$. Here we use a two-layer graph convolutional network (GCN) (Kipf & Welling, 2017) to predict the parameters of the Normal distribution. It is clear that the decoder is invariant to permutations of users.

5. Learning with Monte Carlo Gradients

We have a hybrid latent space since the user embedding is continuous and the network structure A and the interaction type C are discrete. Learning such VAEs typically requires a careful design of Monte Carlo gradient estimation methods (Mohamed et al., 2020), thus being challenging. Although C is only binary, we know from Section 4.2 that it has a strong impact on the AUC of the inferred network structure. It is thus important to accurately infer C to achieve the dominant prior. However, we found the end-to-end training of our model, *i.e.*, minimizing the negative ELBO in Eq. (6), often leads to poor prediction of C. To overcome the difficulty, we exploit a stage-wise training strategy, *i.e.*, 1) pretraining of spectral interaction encoder $q_\phi(C|X)$; 2) training the model with $q_\phi(C|X)$ being fixed. A comparison of end-to-end training and stage-wise training can be found in Appendix D.5.

Pretraining of Spectral Interaction Encoder. In the first stage, we generate a set of networks A from random graph models like Barabási-Albert (BA), ER, and Watts-Strogatz (WS). Relying on these networks, we then simulate equilibrium actions X from the three aforementioned network games. We also collect the ground truth C by checking if the adjacency matrix is more similar to the correlation matrix ψ_{corr} or the anticorrelation matrix $1 - \psi_{\text{corr}}$. We can only pretrain this encoder on synthetic datasets since we do not have ground truth annotation of interaction type in real-world data. The objective function of this stage is,

$$\min_{\phi} \quad \mathbb{E}_{p_{\text{syn}}(X)} \left[\text{KL}(p_{\text{syn}}(C|X) \| q_\phi(C|X))\right], \tag{11}$$

where $p_{\text{syn}}(X)$ is the distribution of equilibrium actions simulated from the previous network games and $p_{\text{syn}}(C|X)$ is a *delta distribution* centered on the ground truth C corresponding to a given X. We can compare Eq. (11) with the term $\mathbb{E}_{p(X)}\left[\text{KL}(q_\phi(C|X) \| p(C))\right]$ in Eq. (6) where we explicitly add the expectation w.r.t. $p(X)$ in ELBO to highlight the difference. The former, *i.e.*, Eq. (11), aims at matching the conditional distribution (*i.e.*, the classifier) and the delta distribution $p_{\text{syn}}(C|X)$. The latter aims at matching the aggregated posterior $\mathbb{E}_{p(X)}\left[q_\phi(C|X)\right]$ with the prior $p(C)$, thus acting as a regularizer in learning the encoder. Therefore, Eq. (11) encourages the encoder to be a good classifier, which is crucial towards our goal (*i.e.*, inferring the latent network structure A) as shown in Section 4.2 and 4.3. We provide more training details in Appendix B.

Training with fixed Spectral Interaction Encoder. In the second stage, we fix the pretrained spectral interaction encoder and train the model by minimizing the negative ELBO in Eq. (6). In particular, we need to compute the gradient of the reconstruction loss w.r.t. the encoder parameters ϕ, *i.e.*, $\nabla_\phi \mathbb{E}_{q_\phi(A,Z|X)}[\log p_\theta(X|A, Z)]$. For the ease of no-

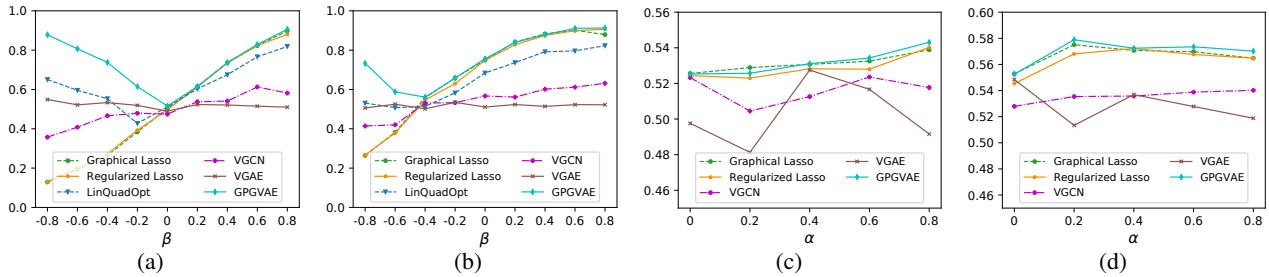

Figure 4. The AUC (y-axis) results on synthetic network games. (a) linear quadratic games with $\alpha = 0.0$; (b) linear quadratic games with $\alpha = 0.8$; (c) linear influence games with BA graphs; (d) linear influence games with ER graphs.

tation, let us denote $R(\phi) := \mathbb{E}_{q_\phi(A,Z|X)}\left[\log p_\theta(X|A,Z)\right]$. To compute the gradient corresponding to the continuous latent variable Z, the pathwise estimator, a.k.a., reparameterization trick (Kingma & Welling, 2014), is often adopted since it is unbiased and low-variance. Specifically, we have $Z = g(\epsilon;\phi_z)$ and $\epsilon \sim p(\epsilon)$, where $g(\epsilon;\phi_z)$ is a sampling path and $p(\epsilon)$ is a base distribution. For discrete latent variable A, we have several choices. 1) The pathwise estimator applied to its continuous relaxation, *e.g.*, binary Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017):

$$A_{ij} = \text{Sigmoid}\left((\log\alpha + \log U - \log(1-U))/\tau\right),$$

where $\log\alpha$ is the logit, $U \sim \text{Uniform}(0,1)$ and temperature $\tau \in (0,\infty)$. 2) Score function estimator, a.k.a., REINFORCE (Williams, 1992). We have,

$$\nabla_\phi R(\phi) \approx \frac{1}{S}\sum_{s=1}^{S} \log p_\theta(X|A^{(s)},Z^{(s)})\nabla_\phi \log q_\phi(A^{(s)}|X),$$

where $A^{(s)} \sim q_\phi(A|X)$, $Z^{(s)} = g(\epsilon^{(s)};\phi_z)$, and $\epsilon^{(s)} \sim p(\epsilon)$. 3) Measure-valued estimator (Heidergott & Vázquez-Abad, 2000; Mohamed et al., 2020). For the mixture of Bernoulli distribution in Eq. (10), recall $\phi = \{\theta_{k,ij}, \alpha_k | k \in [K], i,j \in \mathcal{V}\}$. The measure-valued gradient w.r.t. $\theta_{k,ij}$ is,

$$\nabla_{\theta_{k,ij}} R(\phi) \approx \frac{\alpha_k}{S}\sum_{s=1}^{S} \log p_\theta(X|\{A^{(s)}_{mn}\}, A^{(s)}_{ij}=1, Z^{(s)}) - \frac{\alpha_k}{S'}\sum_{s'=1}^{S'} \log p_\theta(X|\{A^{(s')}_{mn}\}, A^{(s')}_{ij}=0, Z^{(s')}),$$

where $\{A^{(s)}_{mn}\}$ means the set of all $A^{(s)}_{mn}$ such that $(m,n) \neq (i,j)$ and $A^{(s)}_{mn}, A^{(s')}_{mn} \sim \text{Ber}(A_{mn}|\theta_{k,mn})$. The measure-valued gradient w.r.t. α_k as,

$$\nabla_{\alpha_k} R(\phi) \approx \frac{1}{S}\sum_{s=1}^{S} \log p_\theta(X|A^{(s)},Z^{(s)})$$

where $A^{(s)} \sim \prod_{ij}\text{Ber}(A_{ij}|\theta_{k,ij})$.

The pathwise estimator in the discrete case is biased but often has a low variance. The score function estimator is unbiased but typically has high variance, thus often being used together with variance-reduction techniques like the control variate. The measure-valued estimator is mostly unbiased and low-variance, but its high computational costs limit its use in high-dimension problems. We show the derivation of gradients under all these methods in Appendix C. We systematically investigate them in the context of our model and show the experimental results in Section 6.1.

6. Experiments

In this section, we first examine GPGVAE's performance on synthetic network games. Since the observed actions may not satisfy the Nash equilibrium condition, we also evaluate our model on real-world datasets. More details about the experimental setting are provided in Appendix D.2. Code and data are available at `https://github.com/xueyu-ubc/GPGVAE`.

Baselines. We compare GPGVAE with several competitive methods, including optimization based ones like Graphical Lasso (Friedman et al., 2008), Regularized Lasso (Lake & Tenenbaum, 2010), LinQuadOpt (Leng et al., 2020) and BlockRegression (Barik & Honorio, 2019), and learning based ones like VGAE (Kipf & Welling, 2016), DAG-GNN (Yu et al., 2019), VGCN (Elinas et al., 2020), LDS (Franceschi et al., 2019), uGLAD (Shrivastava et al., 2022) and NodeFormer (Wu et al., 2022). Here LinQuadOpt is designed for linear quadratic games, and BlockRegression is proposed under BH graphical games. Although DAG-GNN is proposed to learn DAGs, we still conduct a comparison with it on real-world datasets. Note that most supervised and semi-supervised methods (e.g., NodeFormer) combine graph learning with downstream tasks, e.g., node classification, requiring node labels or partial network connections for training. Such methods cannot be directly applied to our setting and thus we integrate the graph learning part of them with VAEs. Specifically, without node label information, we can use a decoder to reconstruct user actions and compute the reconstruction loss instead of cross-entropy loss.

6.1. Synthetic Data

Data Generation. We first construct three random networks: BA, ER, and WS graphs. Based on these networks,

Table 1. Results on real-world datasets. The average AUC and std for Indian villages are obtained from 48 graphs, and the results for Foursquare and Yelp datasets are based on 3 repetitions.

	Indian Villages	Foursquare		Yelp			
	Villages	New York	São Paulo	PA ratings	PA reviews	LA ratings	LA reviews
Correlation	56.67 ±2.43	52.61	50.77	60.14	60.21	59.24	59.28
Anticorrelation	43.33 ±2.43	47.39	49.23	39.86	39.79	40.76	40.72
Graphical Lasso	**56.91** ±2.19	56.13	56.56	71.49	69.79	66.68	72.57
Regularized Lasso	56.65 ±2.44	55.30	52.48	62.04	56.29	62.45	62.77
BlockRegression	54.67 ±1.66	55.80	52.79	54.22	55.21	56.96	55.90
LinQuadOpt	56.50 ±2.41	54.05	52.58	73.62	75.97	70.47	73.26
uGLAD	52.38 ±2.99	52.07 ±1.53	54.47 ±5.17	69.24 ±3.79	70.84 ±1.55	56.62 ±5.35	70.30 ±4.16
DAG-GNN	52.94 ±1.96	52.24 ±0.80	51.57 ±0.13	69.56 ±0.24	69.67 ±0.64	70.02 ±0.52	69.31 ±0.42
VGAE	50.59 ±3.76	50.54 ±0.90	50.81 ±0.85	53.21 ±2.01	54.18 ±4.12	54.20 ±0.87	53.30 ±0.54
VGCN	53.89 ±1.98	53.39 ±0.11	54.46 ±0.67	62.50 ±0.59	59.63 ±0.97	56.17 ±0.88	57.43 ±1.62
LDS	51.60 ±0.86	54.70 ±0.01	53.38 ±0.01	50.04 ±0.02	50.03 ±0.03	50.09 ±0.02	50.08 ±0.01
NodeFormer	50.56 ±0.37	54.78 ±0.01	53.63 ±0.02	50.08 ±0.00	50.09 ±0.00	50.13 ±0.01	50.14 ±0.00
GPGVAE	*56.77* ±3.07	**57.57** ±0.29	**56.88** ±0.47	**75.35** ±0.67	**76.70** ±1.29	**75.25** ±0.74	**75.08** ±0.22

we generate marginal benefits b from the Gaussian distribution (3) with different parameters α. Then, we simulate equilibrium actions from their distribution. For BH graphical games, the equilibrium actions x^* is the leading eigenvector of $\tilde{A}$, and we consider noisy pure-strategy Nash equilibrium actions, *i.e.*, $x = x^* + e$, where e are independently generated from a sub-Gaussian distribution. The number of nodes in graphs is $N = 20$, and the number of games is $M = 50$.

Table 2. Results for BH graphical games on three random graphs.

	BA graphs	WS graphs	ER graphs
Graphical Lasso	67.57 ±4.60	74.12 ±3.49	58.57 ±3.67
Regularized Lasso	67.59 ±3.93	72.83 ±4.35	58.35 ±3.64
BlockRegression	65.58 ±3.67	72.37 ±3.37	56.35 ±3.27
VGCN	49.77 ±4.97	52.32 ±3.12	50.58 ±4.31
VGAE	55.43 ±4.89	51.18 ±3.00	49.07 ±3.28
GPGVAE	**68.02** ±5.13	**74.45** ±3.45	**58.80** ±3.45

Results. Figure 4 shows the results on linear quadratic games with BA graphs and linear influence games with both BA and ER graphs under different settings. For linear quadratic games, β varies from -0.8 to 0.8, indicating the interactions between individuals shift from strategic substitute to strategic complement. The y-axis denotes the average AUC value across 10 random networks. Specifically, learning-based methods such as VGAE and VGCN exhibit consistently weak performances across various settings, indicating that a non-informative prior over network structure may impede effective learning. Except for LinQuadOpt, optimization-based baselines perform well only in scenarios of strategic complements and they struggle to accurately infer network structure in situations of strategic substitutes. While LinQuadOpt is designed for linear quadratic games and outperforms other baseline methods when $\beta < 0$, it is still inferior to our GPGVAE. Notably, as $\beta \to -0.8$ in Figure 4 (a) and (b), our method's lead over competing approaches becomes more significant. In short, our GPGVAE performs well under both strategic complement and strategic substitute scenarios on these two games, thus verifying the effectiveness of our proposed framework. The averaged AUC and std results on BH graphical games are reported in Table 2. Our GPGVAE consistently outperforms other competitors on BH games across all random networks.

6.2. Ablation Study

We show the ablation study of different gradient estimators on various games with BA graphs in Table 3, where comp. and sub. stand for strategic complement and substitute respectively. We use the exponential moving average (EMA) based control variate technique to reduce the variance of the score function, denoted as score function*. The measure-valued gradient estimator usually has a low variance. However, since it requires sampling in computing the derivative of individual parameters, it does not scale up to large graphs. Therefore, considering both the performance and the computational time, we choose the score function estimator in our experiments. We also study the effect of the number of Bernoulli mixtures and analyze the complexity of our algorithm in Appendix D.5.

6.3. Real-World Data

We choose three real-world datasets. First, the *Indian Villages* dataset consists of rural villages in the southern Indian state of Karnataka. Following the work of Leng et al. (2020; 2023c), we consider 48 villages with ground truth networks. Each node represents a household and the actions

Table 3. Comparison (AUCs) of gradient estimators on linear quadratic games with parameters $\alpha = 0.5, \beta = -0.8$, and 0.8, linear influence games with $\alpha = 0.5$ and BH graphical games.

	Quadratic (sub.)	Quadratic (comp.)	Influence (comp.)	BH (comp.)
Pathwise	87.89 $_{\pm 6.13}$	93.40 $_{\pm 3.55}$	60.64 $_{\pm 8.44}$	82.53 $_{\pm 6.38}$
Score function*	88.12 $_{\pm 5.37}$	93.32 $_{\pm 3.28}$	**62.72** $_{\pm 5.79}$	**82.66** $_{\pm 7.57}$
Measure-valued	**88.22** $_{\pm 6.40}$	**93.75** $_{\pm 3.60}$	62.40 $_{\pm 5.42}$	82.45 $_{\pm 5.87}$

include the number of rooms, the presence of electricity, and other properties. When villagers make decisions related to households, they are often influenced by neighbors. Hence, their actions provide essential information for analyzing the underlying relationships among households. Second, *Foursquare* is a social platform that can help users share locations and find recommendations for various places. When people decide which place to visit or post a review, they are often influenced by others. Therefore, we can infer the social relationships among users based on their actions. We consider two cities, New York and São Paulo for this dataset. For each city, we calculate the number of check-ins as actions. At last, we consider data in Pennsylvania (PA) and Louisiana (LA) states from the social platform *Yelp*, where users can write reviews and recommendations for various businesses. When users make decisions, they are influenced not only by the ratings but also by the content of other users' reviews. Therefore, we use such information as actions to infer their relationships. More details can be found in Appendix D.3.

Results. Table 1 reports the mean and standard deviation of AUC on different datasets. For optimization-based methods like Graphical Lasso, the results are deterministic for different repetitions and we didn't report their std on Foursquare and Yelp datasets. GPGVAE demonstrates superior performance on six out of seven datasets. On the Indian villages dataset, GPGVAE is the second best and slightly worse than the Graphical Lasso. This might be caused by the fact that this dataset only contains 10 games per graph, thus providing little data for learning. Besides, GPGVAE achieves a significant improvement of 2.57% on the New York dataset and 6.78% on the LA (ratings) dataset. The interactions between individuals is predicted as strategic complements, which is consistent with reality. For example, on social platforms, users tend to follow social norms and make choices similar to those around them. This is also supported by the results of correlation and anticorrelation methods in Table 1.

7. Conclusion

In this paper, we introduce GAGVAE, an unsupervised learning framework that infers individuals' interaction types and the underlying network structure based on observed actions from network games. Unlike previous work, our method does not require prior knowledge about specific utility function forms or partial network connections. Our model consists of a data-dependent gated prior, a spectral GNN based interaction type encoder, a Transformer based network structure encoder, and a GNN based action decoder. Extensive experiments on synthetic and real-world datasets verify the effectiveness of our approach. Future works include integrating more expressive deep learning models and extending the approach to large and dynamic networks.

Acknowledgements

We thank Prof. Yifan Sun and Donglin Yang for their valuable feedback and suggestions. We thank the anonymous reviewers for their helpful comments. This work was funded, in part, by the National Nature Science Foundation of China (12171479), the MOE Project of Key Research Institute of Humanities and Social Sciences (NO. 22JJD110001), NSERC DG Grants (No. RGPIN-2022-04636), the Vector Institute for AI, and Canada CIFAR AI Chair. Y.L. acknowledges the support provided by the National Science Foundation [Grant IIS-2153468]. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through the Digital Research Alliance of Canada `alliance.can.ca`, and companies sponsoring the Vector Institute `www.vectorinstitute.ai/#partners`, and Advanced Research Computing at the University of British Columbia. Additional hardware support was provided by John R. Evans Leaders Fund CFI grant.

Impact Statement

This paper presents a valuable technique to capture major interaction relationships among users, such as strategic complement and strategic substitute. The proposed unsupervised learning framework can effectively infer network structures from observed user actions, which is particularly useful when connections between users are difficult to access or private. The insights derived from the inferred network structures are also beneficial for resource optimization and information dissemination. Our work will be instrumental to researchers working on network games and graph-related research.

References

Aral, S., Muchnik, L., and Sundararajan, A. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *Proceedings of the National Academy of Sciences*, 106(51):21544–21549, 2009.

Ballester, C., Calvó-Armengol, A., and Zenou, Y. Who's who in networks. wanted: The key player. *Econometrica*, 74(5):1403–1417, 2006.

Banerjee, A., Chandrasekhar, A. G., Duflo, E., and Jackson, M. O. The diffusion of microfinance. *Science*, 341(6144): 1236498, 2013.

Banerjee, O., El Ghaoui, L., and d'Aspremont, A. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *Journal of Machine Learning Research*, 9:485–516, 2008.

Barik, A. and Honorio, J. Provable computational and statistical guarantees for efficient learning of continuous-action graphical games. In *CoRR*, 2019.

Belilovsky, E., Kastner, K., Varoquaux, G., and Blaschko, M. B. Learning to discover sparse graphical models. In *ICML*, 2016.

Bo, D., Shi, C., Wang, L., and Liao, R. Specformer: Spectral graph neural networks meet transformers. In *ICLR*, 2023.

Bramoullé, Y., Kranton, R., and D'Amours, M. Strategic interaction and networks. *American Economic Review*, 104(3):898–930, 2014.

Bulow, J. I., Geanakoplos, J. D., and Klemperer, P. D. Multimarket oligopoly: Strategic substitutes and complements. *Journal of Political Economy*, 93(3):488–511, 1985.

Centola, D. and Macy, M. Complex contagions and the weakness of long ties. *American journal of Sociology*, 113(3):702–734, 2007.

Diamond, S. and Boyd, S. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Dong, X., Thanou, D., Toni, L., Bronstein, M., and Frossard, P. Graph signal processing for machine learning: A review and new perspectives. *IEEE Signal processing magazine*, 37(6):117–127, 2020.

Elinas, P., Bonilla, E. V., and Tiao, L. Variational inference for graph convolutional networks in the absence of graph data and adversarial settings. In *NeurIPS*, 2020.

Franceschi, L., Niepert, M., Pontil, M., and He, X. Learning discrete structures for graph neural networks. In *ICML*, 2019.

Friedman, J., Hastie, T., and Tibshirani, R. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

Galeotti, A., Golub, B., and Goyal, S. Targeting interventions in networks. *Econometrica*, 88(6):2445–2471, 2020.

Ghoshal, A. and Honorio, J. From behavior to sparse graphical games: Efficient recovery of equilibria. In *Allerton*, 2016.

Goyal, S. and Moraga-González, J. L. R&d networks. *The RAND Journal of Economics*, 32(4):686–707, 2001.

Heidergott, B. and Vázquez-Abad, F. J. Measure valued differentiation for stochastic processes: The finite horizon case. *Journal of Optimization Theory and Applications*, 136:187–209, 2000.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2016.

Honorio, J. and Ortiz, L. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research*, 16(1):1157–1210, 2015.

Irfan, M. T. and Ortiz, L. E. A game-theoretic approach to influence in networks. In *AAAI*, 2011.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *ICLR*, 2014.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. In *ICML*, 2018.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Lake, B. and Tenenbaum, J. Discovering structure by learning sparse graphs. In *CogSci*, 2010.

Leng, Y., Dong, X., Wu, J., and Pentland, A. Learning quadratic games on networks. In *ICML*, 2020.

Leng, Y., Chen, Y., Dong, X., Wu, J., and Shi, G. Privacy risks of social interaction structure: Network learning in quadratic games. *Available at SSRN 3875878*, 2023a.

Leng, Y., Dong, X., Moro, E., and Pentland, A. Long range effect of social influence via phone communication networks. *Information Systems Research*, 2023b.

Leng, Y., Sowrirajan, T., Zhai, Y., and Pentland, A. Interpretable stochastic block influence model: measuring social influence among homophilous communities. *IEEE Transactions on Knowledge and Data Engineering*, 2023c.

Lim, D., Robinson, J., Jegelka, S., and Maron, H. Expressive sign equivariant networks for spectral geometric learning. *arXiv preprint arXiv:2312.02339*, 2023.

Liu, Z., Yang, D., Wang, Y., Lu, M., and Li, R. Egnn: Graph structure learning based on evolutionary computation helps more in graph neural networks. *Applied Soft Computing*, 135:110040, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

Meinshausen, N. and Bühlmann, P. High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, 34(3):1436–1462, 2006.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(1):5183–5244, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Pu, X., Cao, T., Zhang, X., Dong, X., and Chen, S. Learning to learn graph topologies. In *NeurIPS*, 2021.

Rossi, E., Monti, F., Leng, Y., Bronstein, M., and Dong, X. Learning to Infer Structures of Network Games. In *ICML*, 2022.

Shrivastava, H., Chen, X., Chen, B., Lan, G., Aluru, S., Liu, H., and Song, L. Glad: Learning sparse graph recovery. In *ICLR*, 2020.

Shrivastava, H., Chajewska, U., Abraham, R., and Chen, X. A deep learning approach to recover conditional independence graphs. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Wang, H., Fu, Y., Yu, T., Hu, L., Jiang, W., and Pu, S. Prose: Graph structure learning via progressive strategy. In *SIGKDD*, 2023.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Wu, Q., Zhao, W., Li, Z., Wipf, D., and Yan, J. Nodeformer: A scalable graph structure learning transformer for node classification. In *NeurIPS*, 2022.

Xu, L., Chen, L., Wang, R., Nie, F., and Li, X. Joint feature and differentiable k-NN graph learning using dirichlet energy. In *NeurIPS*, 2023.

Yu, Y., Chen, J., Gao, T., and Yu, M. DAG-GNN: DAG structure learning with graph neural networks. In *ICML*, 2019.

Zhang, G., Hu, Z., Wen, G., Ma, J., and Zhu, X. Dynamic graph convolutional networks by semi-supervised contrastive learning. *Pattern Recognition*, 139:109486, 2023.

Zhao, J., Wen, Q., Ju, M., Zhang, C., and Ye, Y. Self-supervised graph structure refinement for graph neural networks. In *WSDM*, 2023.

Zou, D., Peng, H., Huang, X., Yang, R., Li, J., Wu, J., Liu, C., and Yu, P. S. Se-gsl: A general and effective graph structure learning framework through structural entropy optimization. *arXiv preprint arXiv:2303.09778*, 2023.

A. Network Games and Dominant Priors

A.1. Common Network Games

Table 4 provides detailed information on the three network games introduced in the paper, where **u** represents the eigenvector associated with the eigenvalue 1 of $\tilde{A}$. We use linear quadratic games as an example to derive the explicit solution for

Table 4. Three common network games.

	Utility function u_i	Nash equilibrium x^*	Distribution of x^*
BH graphical games	$\sum_{j\in\mathcal{N}_i} \tilde{A}_{ij}x_ix_j - b_ix_i$	$\mathbf{u}$	–
Linear influence	$\sum_{j\in\mathcal{N}_i} \tilde{A}_{ij}x_ix_j - b_ix_i$	$\tilde{A}^{-1}b$	$\mathcal{N}(0, \tilde{A}^{-1}(I-\alpha\tilde{A})^{\dagger}\tilde{A}^{-1})$
Linear quadratic	$b_ix_i - \frac{1}{2}x_i^2 + \beta\sum_{j\in\mathcal{N}_i} \tilde{A}_{ij}x_ix_j$	$(I-\beta\tilde{A})^{-1}b$	$\mathcal{N}(0, (I-\beta\tilde{A})^{-1}(I-\alpha\tilde{A})^{\dagger}(I-\beta\tilde{A})^{-1})$

equilibrium action $x^* = [x_1^*, ..., x_N^*]^\top$ and its distribution. Remember the utility function of linear quadratic games is

$$u_i = b_ix_i - \frac{1}{2}x_i^2 + \beta\sum_{j\in\mathcal{N}_i}\tilde{A}_{ij}x_ix_j.$$

Computing the first-order derivative of the payoff u_i with respect to action x_i, we have

$$\frac{\partial u_i}{\partial x_i} = b_i - x_i + \beta\sum_{j\in\mathcal{N}_i}\tilde{A}_{ij}x_j, \quad i\in\mathcal{V}.$$

Considering all individuals $i \in \mathcal{V}$ simultaneously, we can obtain the Nash equilibrium action as follows

$$x^* = (I-\beta\tilde{A})^{-1}b,$$

Combined with the distribution of marginal benefit b (3), the Nash equilibrium action x^* obeys a multivariate Gaussian distribution

$$x^* \sim \mathcal{N}(0, (I-\beta\tilde{A})^{-1}(I-\alpha\tilde{A})^{\dagger}(I-\beta\tilde{A})^{-1}).$$

A.2. Dominant Priors in Different Games

In this paper, we consider two types of priors: Correlation and Anticorrelation. The term "dominant prior" refers to the prior that is more similar to the ground truth, *i.e.*, has a higher AUC value. To confirm the type of dominant prior in different network games, we first generate 50 random graphs and then derive equilibrium behaviors based on their distribution under different games. We calculate the AUC values of both correlation and anticorrelation priors on each graph. If the AUC value of correlation surpasses that of anticorrelation, we consider correlation prior dominants that network; conversely, anticorrelation is considered dominant. As a result, we can obtain the proportion of each prior across the 50 random networks.

The distribution of dominant priors under linear quadratic games on ER graphs is provided in Figure 2. Table 5 reports the proportion under linear influence games and BH graphical games on BA, ER, and WS graphs. For linear influence games, we consider three cases, $\alpha = 0, 0.5$, and 0.8. When $\alpha > 0$, the marginal benefit vector b is "smooth" on networks, meaning that individuals b_i and b_j's "thresholds" are closer if i and j are connected, relative to when i and j are independent. As we can see, when $\alpha = 0$, there are a few cases where the dominant priors are anticorrelation. However, as α increases, all of the networks are dominant by correlation prior. For BH graphical games, the dominant priors are always correlation prior.

Table 5. Proportion of dominant priors under linear influence games and BH graphical games (correlation/anticorrelation).

	Linear influence ($\alpha = 0$)	Linear influence ($\alpha = 0.5$)	Linear influence ($\alpha = 0.8$)	BH
BA graphs	46/4	47/3	50/0	50/0
ER graphs	49/1	50/0	50/0	50/0
WS graphs	48/2	49/1	50/0	50/0

B. Model Architecture

Pretraining of Spectral Interaction Encoder. In the first stage, we generate $2,000$ random networks on BA, ER, and WS graphs. The number of nodes is uniformly sampled from the range $[10, 50]$. For each network, we randomly choose one type of game from linear quadratic games, linear influence games, and BH graphical games. After this, we generate 100 equilibrium actions in an i.i.d. manner based on their distribution. For both linear quadratic games and linear influence games, the parameter $\alpha \in [0, 1]$, and $\beta \in [-0.9, 0.9]$ for linear quadratic games. For BH graphical games, we consider noisy pure-strategy equilibrium actions with parameter $\epsilon = 0.2$. For each graph, the ground truth C is determined by comparing the AUC value of the correlation matrix and anticorrelation matrix with the adjacency matrix of random networks. Then, we use a spectral GNN, *i.e.*, Specformer (Bo et al., 2023), to model encoder $p_\theta(C|X)$. Below, we outline the detailed steps.

(1) Conduct eigendecomposition of covariance $\text{cov}(X) = U\Lambda U^\top$, where $\Lambda = \text{diag}(\lambda_1, ..., \lambda_N)$.

(2) Eigenvalue Encoding: map each eigenvalue $\lambda_i, i = 1, , , .N$ to a high-dimensional vector as follows:

$$\rho(\lambda, 2j) = \sin(\epsilon\lambda/10000^{2j/d}),$$
$$\rho(\lambda, 2j+1) = \cos(\epsilon\lambda/10000^{2j/d}),$$

where d is a hyperparameter that determines the dimension of the representation space, j represents the position of an element, and ϵ is a hyperparameter used to control the influence of eigenvalues. We set $d = 64$ and $\epsilon = 100$ as suggested by the original paper. The initial representation of the eigenvalues is obtained by concatenating the eigenvalues and the corresponding encodings, *i.e.*, $Z = [\lambda_1||\rho(\lambda_1), ..., \lambda_N||\rho(\lambda_N)]^\top$, and then apply multiple Transformer blocks to obtain new representations.

(3) Eigenvalue Decoding: Z_l is the representation obtained by the l-th attention head. By performing spectral filtering $\bar{\lambda}_l = \phi(Z_l W)$, where ϕ is the activation and W is the weight parameter, we get new eigenvalues $\bar{\lambda}_1, ..., \bar{\lambda}_L$ ($\bar{\lambda}_l \in \mathbb{R}^{N\times 1}$) and new bases $S_l = U\text{diag}(\bar{\lambda}_l)U^\top \in \mathbb{R}^{N\times N}$, where U is the eigenvector of $\text{cov}(X)$. These bases are connected $S = [I_N||S_1||\cdots||S_L]$ and fed to a feed-forward network to obtain learnable bases $\hat{S}$. After multiple graph convolutional layers, the final sigmoid layer outputs the probability for each type.

Details of User Embedding Encoder. For latent user embedding Z, the posterior is

$$q_\phi(Z|X) = \prod_{i=1}^{N} q_\phi(Z_i|X_i) = \prod_{i=1}^{N} \mathcal{N}(Z_i; \mu_i, \text{diag}(\sigma_i^2)).$$

We use a fully connected feed-forward network to learn the mean matrix $\mu \in \mathbb{R}^{N\times D}$, *i.e.*, $\mu = \text{FFN}(X)$, where $\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$. The dimension $D = 256$, and the variance is fixed as $\sigma^2 = 1$.

Details of Network Structure Encoder. Based on the underlying network structure, the behavior of different individuals can be influenced by their neighbors. We utilize a Transformer-like (Vaswani et al., 2017) architecture to model encoder $q_\phi(A|X)$ and the encoder performs the following message-passing operations:

$$\begin{aligned} h &= \text{Transformer}(X), \\ s_{ij} &= \frac{1}{2}[\text{FFN}(\text{LN}(h_i - h_j)) + \text{FFN}(\text{LN}(h_j - h_i))], \\ \theta_{1,ij}, ..., \theta_{K,ij} &= \psi(\text{Sigmoid}(\text{FFN}(s_{ij}))), \\ \alpha_1, ..., \alpha_K &= \text{Softmax}(\square\text{FFN}(s_{ij})), \end{aligned}$$

where LN denotes layer normalization. Function $\psi(x) := \frac{1}{2}(x + x^\top)$ is used to ensure symmetry. $\square$ denotes the mean operation. We first get $h = [h_1^\top, ..., h_N^\top]^\top$ based on a Transformer model, and h_i is a vector of F features. By performing a pairwise operation on h, we can get a hidden vector $s_{ij} \in \mathbb{R}^F$ for each pair of nodes (i, j), and finally compute the probability α_k of each component and the corresponding parameter $\theta_{k,ij}$. The number of attention heads in Transformer is 4 and the dimensionality of inner layers is $F = 256$. Similar to β-VAE (Higgins et al., 2016), we use a constant δ to control the distance between the posterior $q_\phi(A|X)$ and the gated-prior $p(A|C, X')$ while balancing the KL divergence of the graph encoder and the reconstruction quality.

Details of Decoder. For decoder $p_\theta(X|A, Z)$, we assume conditional independence among different users, *i.e.*, $p_\theta(X|A, Z) = \prod_{i=1}^{N} p_\theta(X_i|A, Z_i) = \prod_{i=1}^{N} \mathcal{N}(X_i; \mu_i, \text{diag}(\sigma_i^2))$. We use a two-layer GCN to predict the mean ma-

trix $\mu = \text{GCN}(A, Z)$ and the variance is $\sigma^2 = 1$. The two-layer GCN is defined as $\text{GCN}(A, Z) = \tilde{A}\text{ReLU}(\tilde{A}ZW_0)W_1$, with a nonlinear function $\text{ReLU}(\cdot) = \max(0, \cdot)$ and $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. W_0 and W_1 are weight parameters.

Permutation Invariant. For the user embedding encoder, we consider a conditionally independence posterior distribution, and use MLP to model it, thus this encoder is permutation invariant to users. For the network structure encoder, we consider the Transformer without the positional encoding layer, which can guarantee the permutation equivariant to users, and this encoder is also invariant to users. For the spectral GNN, we use the covariance matrix $\text{cov}(X)$, and its eigendecomposition is permutation invariant. The ground truth C is determined by the AUC values of correlation and anticorrelation priors, which is permutation invariant to users. The Specformer model we used is also permutation equivariant. Besides, we use GCN to model the decoder and it's also invariant to permutations of users.

C. Gradient Estimators

Consider the following form of the negative reconstruction function for Monte Carlo estimation:

$$\mathbb{E}_{q_\phi(A,Z|X)}[\log p_\theta(X|A, Z)] \approx \frac{1}{S}\sum_{s=1}^{S}[\log p_\theta(X|A^{(s)}, Z^{(s)})],$$

where $A^{(s)} \sim q_\phi(A|X)$ and $Z^{(s)} \sim q_\phi(Z|X)$. The gradient of the expectation $\mathbb{E}_{q_\phi(A,Z|X)}[\log p_\theta(X|A, Z)]$ with respect to ϕ is intractable. In this section, we provide details of three Monte Carlo gradient estimators: pathwise, score function, and measure-valued gradient estimators.

C.1. Pathwise Estimator

For continuous latent variable Z, we can use the reparameterization trick. Specifically, we can first sample a random variable ϵ from a standard Gaussian distribution $p(\epsilon) = \mathcal{N}(\epsilon; 0, I)$, and then reparameterize z_i as a deterministic function of ϵ: $z_i = g(\epsilon; \phi_z) = \mu_i + \sigma_i\epsilon$ with learned parameters $\phi_z = \{\mu_i, \sigma_i\}$.

For the discrete graph A, we can consider the binary Gumbel-softmax trick. Each edge A_{ij} is estimated by:

$$\text{Sigmoid}(\frac{\log\alpha + \log U - \log(1-U)}{\tau}),$$

where $\log\alpha$ is the logit output, $U \sim \text{Uniform}(0, 1)$, and $\log U - \log(1-U)$ is a Logistic random variable. The temperature hyperparameter $\tau \in (0, \infty)$ controls the smoothness of the sampling process and the variance of the gradients. The Gumbel-softmax is a continuous relaxation of discrete distributions, thus it is a biased gradient estimator. In this paper, we use the reparameterization trick to sample latent representation Z from posterior $p_{\phi_z}(Z|X)$ and choose the temperature $\tau = 0.5$ when using the binary Gumbel-softmax.

C.2. Score Function Estimator

The score function is the derivative of the log probability of the distribution $\log q_\phi(x)$ to its parameter ϕ:

$$\nabla_\phi \log q_\phi(x) = \frac{\nabla_\phi q_\phi(x)}{q_\phi(x)}.$$

For convenience, we use the reparameterization trick for continuous variable $Z = g(\epsilon; \phi_z)$. Then the derivative of the expectation with respect to the parameter ϕ_A of the graph encoder is

$$\begin{aligned}
\nabla_{\phi_A}\mathbb{E}_{q_\phi(A,Z|X)}\log p_\theta(X|A, Z) &= \nabla_{\phi_A}\int q_{\phi_A}(A|X)q(\epsilon)\log p_\theta(X|A, Z = g(\epsilon;\phi_z))\mathrm{d}A\mathrm{d}\epsilon \\
&= \int \nabla_{\phi_A} q_{\phi_A}(A|X)q(\epsilon)\log p_\theta(X|A, Z = g(\epsilon;\phi_z))\mathrm{d}A\mathrm{d}\epsilon \\
&= \int q_{\phi_A}(A|X)\nabla_{\phi_A}\log q_{\phi_A}(A|X)q(\epsilon)\log p_\theta(X|A, Z = g(\epsilon;\phi_z))\mathrm{d}A\mathrm{d}\epsilon \\
&= \mathbb{E}_{q_{\phi_A}(A|X)q(\epsilon)}[\log p_\theta(X|A, Z = g(\epsilon;\phi_z))\nabla_{\phi_A}\log q_{\phi_A}(A|X)] \\
&\approx \frac{1}{S}\sum_{s=1}^{S}\log p_\theta(X|A^{(s)}, Z^{(s)})\nabla_{\phi_A}\log q_{\phi_A}(A^{(s)}|X)
\end{aligned}$$

where $A^{(s)} \sim q_{\phi_A}(A|X)$, $Z^{(s)} = g(\epsilon^{(s)}; \phi_z)$ and $\epsilon^{(s)} \sim p(\epsilon)$. Finally, we can get a Monte Carlo estimator of the gradient $\nabla_{\phi_A} \mathbb{E}_{q_\phi(A,Z|X)} \log p_\theta(X|A, Z)$.

Although the score function is an unbiased estimator of the gradient, it has many variance sources, such as the number of nodes. Usually, it needs some variance reduction techniques such as control variables to reduce the variance. In this paper, we use the exponential moving average (EMA) to control its variance.

C.3. Measure-Valued Gradient Estimator

Consider the derivative of $q(x;\theta)$ with respect to a single parameter θ_i, the measure-valued gradient estimator decomposes the derivative $\nabla_{\theta_i} q(x;\theta)$ into the following form:

$$\nabla_{\theta_i} q(x;\theta) = c_{\theta_i}(q_i^+(x;\theta_i) - q_i^-(x;\theta_i)),$$

where q_i^+ and q_i^- are two densities, c_{θ_i} is a constant. For a univariate Bernoulli distribution, $q_i^+ = \delta_1$ and $q_i^- = \delta_0$ are two Dirac densities at 1 and 0, respectively. The constant $c_{\theta_i} = 1$.

For a single Bernoulli distribution $q_{\phi_A}(A|X) = \prod_{i \le j} \text{Ber}(A_{ij}|\theta_{ij}) = \prod_{i \le j} \theta_{ij}^{A_{ij}}(1-\theta_{ij})^{1-A_{ij}}$ and $\phi_A = \{\theta_{ij}, i, j \in \mathcal{V}\}$, the derivative of the expectation $\mathbb{E}_{q_\phi(A,Z|X)} \log p_\theta(X|A, Z)$ with respect to θ_{ij} is

$$\begin{aligned}
&\nabla_{\theta_{ij}} \mathbb{E}_{q_\phi(A,Z|X)} \log p_\theta(X|A, Z) \\
&= \nabla_{\theta_{ij}} \int \prod_{i \le j} \theta_{ij}^{A_{ij}}(1-\theta_{ij})^{1-A_{ij}} q(\epsilon) \log p_\theta(X|A, Z = g(\epsilon;\phi_z)) \mathrm{d}A\mathrm{d}\epsilon \\
&= \int \nabla_{\theta_{ij}} [\theta_{ij}^{A_{ij}}(1-\theta_{ij})^{1-A_{ij}}] \prod_{m \le n} [\theta_{mn}^{A_{mn}}(1-\theta_{mn})^{1-A_{mn}}] q(\epsilon) \log p_\theta(X|A, Z = g(\epsilon;\phi_z)) \mathrm{d}A\mathrm{d}\epsilon \\
&= \int (\delta_1(A_{ij}) - \delta_0(A_{ij})) \prod_{m \le n} [\theta_{mn}^{A_{mn}}(1-\theta_{mn})^{1-A_{mn}}] q(\epsilon) \log p_\theta(X|A, Z = g(\epsilon;\phi_z)) \mathrm{d}A\mathrm{d}\epsilon \\
&= \mathbb{E}_{\delta_1(A_{ij}) \prod_{m \le n} \text{Ber}(A_{mn}|\theta_{mn}) q(\epsilon)} [\log p_\theta(X|A, Z = g(\epsilon;\phi_z))] \\
&\quad - \mathbb{E}_{\delta_0(A_{ij}) \prod_{m \le n} \text{Ber}(A_{mn}|\theta_{mn}) q(\epsilon)} [\log p_\theta(X|A, Z = g(\epsilon;\phi_z))] \\
&\approx \frac{1}{S} \sum_{s=1}^{S} \log p_\theta(X|\{A_{mn}^{(s)}\}, A_{ij}^{(s)} = 1, Z^{(s)}) - \frac{1}{S'} \sum_{s'=1}^{S'} \log p_\theta(X|\{A_{mn}^{(s')}\}, A_{ij}^{(s')} = 0, Z^{(s')}),
\end{aligned}$$

where $\{A_{mn}^{(s)}\}$ means the set of all $A_{mn}^{(s)}$ such that $(m,n) \neq (i,j)$ and $A_{mn}^{(s)}, A_{mn}^{(s')} \sim \text{Ber}(A_{mn}|\theta_{mn})(m \le n)$. $Z^{(s)} = g(\epsilon^{(s)}; \phi_z)$, $Z^{(s')} = g(\epsilon^{(s')}; \phi_z)$ and $\epsilon^{(s)}, \epsilon^{(s')} \sim p(\epsilon)$.

Similarly, for a mixture of Bernoulli distribution $q_{\phi_A}(A|X) = \sum_k \alpha_k \prod_{i \le j} \theta_{k,ij}^{A_{ij}}(1-\theta_{k,ij})^{1-A_{ij}}$, $\phi_A = \{\theta_{k,ij}, \alpha_k | k \in [K], i, j \in \mathcal{V}\}$. We have the derivative w.r.t. α_k as,

$$\nabla_{\alpha_k} \mathbb{E}_{q_\phi(A,Z|X)} \log p_\theta(X|A, Z) \approx \frac{1}{S} \sum_{s=1}^{S} \log p_\theta(X|A^{(s)}, Z^{(s)})$$

where $A^{(s)} \sim \prod_{i \le j} \text{Ber}(A_{ij}|\theta_{k,ij})$. The derivative w.r.t. $\theta_{k,ij}$ is,

$$\begin{aligned}
\nabla_{\theta_{k,ij}} \mathbb{E}_{q_\phi(A,Z|X)} \log p_\theta(X|A, Z) \approx &\frac{\alpha_k}{S} \sum_{s=1}^{S} \log p_\theta(X|\{A_{mn}^{(s)}\}, A_{ij}^{(s)} = 1, Z^{(s)}) \\
&- \frac{\alpha_k}{S'} \sum_{s'=1}^{S'} \log p_\theta(X|\{A_{mn}^{(s')}\}, A_{ij}^{(s')} = 0, Z^{(s')}),
\end{aligned}$$

where $A_{mn}^{(s)}, A_{mn}^{(s')} \sim \text{Ber}(A_{mn}|\theta_{k,mn})(m \le n)$.

The measure-valued estimator is unbiased in most cases and its variance depends on the choice of the decomposition of the derivative. To evaluate the gradient for each parameter, we need to perform sampling twice. One strategy to reduce the variance is to employ the same sampling distributions for each parameter. However, it's important to note that measure-valued gradient estimators have higher computational costs compared to score-function and pathwise estimators. This increased computational expense arises from the need to compute gradients individually for each parameter, making it

a more resource-intensive process.

D. Simulation Details and Additional Numerical Experiments

D.1. Details of data-dependent priors comparison

To compare the performance of graph VAE based on different priors $p(A)$, we consider linear quadratic games, which contain both strategic complements and strategic substitute interactions. The smoothness parameter $\alpha = 0$ and $\beta \in [-0.8, 0.8]$. We randomly generate 50 ER graphs (20 nodes and 100 games) for each setting, and then generate user actions X based on the distribution of equilibrium actions. For each prior, we train the same VAE model with different priors and compute the AUC value between the learned structure and ground truth. Specifically,

1) *Uniform*. The prior $p(A) = \prod_{i,j} p(A_{ij})$ and $p(A_{ij} = 1) = 1/2$. In the second stage, we train a VAE model, using this prior, to infer the network structure.

2) *Correlation and Anticorrelation*. The prior $p(A) = \prod_{i,j} p(A_{ij})$, $p(A_{ij} = 1) = \psi_{\text{corr}}(X_i, X_j) := (\rho_{ij} + 1)/2$ for correlation and $p(A_{ij} = 1) = 1 - \psi_{\text{corr}}(X_i, X_j) = (1 - \rho_{ij})/2$ for anticorrelation. ρ_{ij} is the Pearson's correlation coefficient between users i and j based on their sampled actions and the definition is provided in Section 4.2. In the second stage, we independently employ two separate VAE models to infer the network structure.

3) *MLP Enc.* We introduce latent variable C to determine whether to use correlation prior ($C = 1$) or anticorrelation prior ($C = 0$). In the first stage, we trained a two-layer MLP model on 2000 different types of random graphs and various network games to learn the encoder $q_\phi(C|X)$. The dimension of the input layer depends on the number of nodes, and the inputs consist of the eigenvalues of the covariance $\text{cov}(X)$. The final layer utilizes a sigmoid activation function to output the probabilities associated with each type. The ground truth C is determined by comparing the AUC value of the correlation matrix and anticorrelation matrix with the adjacency matrix of random networks. In the second stage, we estimated the prior types C on ER graphs based on this pre-trained model. These ER graphs are the same as those used for *Uniform*, *Correlation*, and *Anticorrelation* priors. According to different types C, we adopt different priors $p(A|C, X')$ when training the VAE model. X' is a subset of data X, for example, a subset of games within X.

4) *Spectral Enc.* The pre-training process for the spectral GNN is the same as the MLP encoder except that the model used to learn the encoder $q_\phi(C|X)$ is replaced with Specformer. More details can be found in Appendix B.

For a fair comparison, we use the same VAE model except for the priors. As previously described, the user embedding encoder $q_\phi(Z|X)$ is modeled by a two-layer MLP, and the graph encoder $q_\phi(A|X)$ is modeled by a transformer-like architecture. $q_\phi(A|X)$ is simplified to a single Bernoulli distribution. The decoder $p_\theta(X|A, Z)$ is parameterized by a two-layer GCN. Additionally, we use the pathwise gradient estimator and AdamW optimizer (Loshchilov & Hutter, 2017).

D.2. Experimental setup

Baselines. We provide a brief introduction to the baselines in our work.

- Correlation: It uses the normalized Pearson's correlation coefficient between users i and j based on their sampled actions as the estimated networks, *i.e.*, $(\rho_{X_i,X_j} + 1)/2 \in [0, 1]$
- Anticorrelation: It uses the negative correlation coefficient as the estimated networks, *i.e.*, $(1 - \rho_{X_i,X_j})/2 \in [0, 1]$.
- Graphical Lasso: The presence of zero entries in the inverse covariance matrix of a multivariate normal distribution indicates conditional independence relationships among variables. Thus, this method estimates a sparse graphical model by incorporating a lasso penalty to the inverse covariance matrix.
- Regularized Lasso: Similar to the Graphical Lasso, this method assumes that node features obey a Gaussian distribution and it maximizes a posteriori (MAP) estimate of the graph. The sparsity in the graph structure is achieved by assuming that each edge is independently drawn from an exponential distribution.
- LinQuadOpt: This method is introduced to infer the interaction network from observations within the specific linear quadratic games. They formulate an optimization problem grounded in the established relationship between equilibrium

actions and the network structure. In this paper, we use the algorithm that considers homophilous marginal benefits and it demonstrated superior performance compared to another proposed algorithm in the original paper.

- BlockRegression: This method is proposed to recover the underlying structure of a graphical game using a block regularized method. For the graphical game, they consider noisy pure-strategy Nash equilibrium actions (ε-PSNE), where noise is independently added to the equilibrium actions of each individual.
- uGLAD: This method is an extension of GLAD (Shrivastava et al., 2020), which solves the Graphical Lasso problem through an unsupervised learning approach.
- VGAE: The authors introduce a VAE model aimed at learning interpretable latent representations for graphs. The graph structure is derived through an inner product decoder. In the setting of this paper, where prior knowledge of the graph structure is unknown, we employ a fully connected graph as the input for the VGAE encoder.
- VGCN: This approach employs a joint probabilistic model with a GCN-based likelihood for semi-supervised classification tasks. Similar to our work, it considers the prior distribution and develops a stochastic variational inference algorithm to estimate the posterior distribution of the graph through the concrete distributions. Following their paper, we use k-NN to construct the prior distribution of the graph.
- DAG-GNN: This method is designed to learn the weighted adjacency matrix of a directed acyclic graph. Utilizing a linear structural equation model, it incorporates a VAE model parameterized with graph neural network architectures. While the algorithm is specifically tailored for DAGs, we also assess its performance on real-world datasets.
- LDS: This semi-supervised learning method integrates node classification to simultaneously learn both the graph structure and the parameters of GCNs. It does this by approximately solving a bilevel optimization problem that determines a discrete probability distribution for the graph edges. We integrate the graph learning part with VAEs and use a decoder to reconstruct user actions.
- NodeFormer: This semi-supervised learning method introduces a new class of scalable Transformers for efficient node classification by propagating messages between arbitrary node pairs in flexible layer-specific latent graphs. Without node label information, we integrate the graph learning part with VAEs. Specifically, we use a decoder to reconstruct user actions and compute the reconstruction loss instead of cross-entropy loss.

Probelm Solving. Graphical Lasso can be solved by coordinate descent and Lars algorithm. We use the Scikit-learn GraphicalLasso[1] to solve it. Regularized Lasso, LinQuadOpt, and BlockRegression are solved based on CVXPY (Diamond & Boyd, 2016), a package for solving convex programs, which is not able to compute datasets with a large number of nodes. To conduct experiments on large graphs, *i.e.*, *Yelp*, we implement the algorithm using a gradient-based method to solve the associated convex problems. For uGLAD, LDS, NodeFormer, DAG-GNN, VGAE, and VGCN, we adopt the algorithms provided by the authors and implement them using PyTorch (Paszke et al., 2019). We also implement our method in Pytorch and optimize the AdamW optimizer with a learning rate $1e-4$ and the parameters $\epsilon = 1e-8$ and $(\beta_1, \beta_2) = (0.9, 0.999)$.

Hyperparameter tuning. For optimization algorithms like Graphical Lasso, regularization parameters play a crucial role in determining the sparsity structure of graphs. For learning-based methods such as VGCN, the choice of parameter k is also important in constructing the prior distribution of the graph via k-NN. A common approach for hyperparameter tuning is to select the appropriate parameters with the help of validation sets, but under the unsupervised learning requirements of this paper, we cannot obtain any prior network structure information in advance. Therefore, inspired by GCNs, we design a reconstruction objective function

$$\tilde{\alpha} = \arg\min_{\alpha}\{\min_{W} \|A_\alpha XW - X\|_2^2, \alpha \in \mathcal{S}\}$$

where $\alpha \in \mathcal{S}$ is the hyperparameter, A_α is the network structure obtained under the corresponding hyperparameter α. W is weight parameter and $W = ((AX)^\top AX)^\dagger (AX)^\top X$, where $\dagger$ denotes pseudo-inverse. For each method, we select a feasible parameter range $\mathcal{S}$ based on the guidelines provided in their papers. Then, we conduct a grid search over $\mathcal{S}$ to select the hyperparameter that minimizes the reconstruction loss $\|A_\alpha XW - X\|_2^2$ as the final parameter $\tilde{\alpha}$. Here, the objective function used for tuning hyperparameters is designed to satisfy the following conditions: 1) a convex function that guarantees a unique solution; 2) it can be solved efficiently.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.covariance.GraphicalLasso.html

D.3. Data Generation

Synthetic. We consider three random graphs: ER, WS, and BA graphs. Based on these network structures, we generate Nash equilibrium actions for different network games. For linear quadratic games and linear influence games, we first generate marginal benefits b from the Gaussian distribution (3) with different smoothness parameters $\alpha \in [0, 1]$, and then we simulate equilibrium actions following specific distributions: $x^* \sim \mathcal{N}(0, \tilde{A}^{-1}(I - \alpha\tilde{A})^{\dagger}\tilde{A}^{-1})$ for linear influence games, and $x^* \sim \mathcal{N}(0, (I - \beta\tilde{A})^{-1}(I - \alpha\tilde{A})^{\dagger}(I - \beta\tilde{A})^{-1})$ for linear quadratic games. To guarantee that $(I - \beta\tilde{A})$ is invertible, parameter $|\beta| < 1$. For BH graphical games, the Nash equilibrium actions x^* correspond to the eigenvector of $\tilde{A}$ associated with its largest eigenvalue 1. Following the setting in (Barik & Honorio, 2019), we consider noisy pure-strategy Nash equilibrium actions (ε-PSNE), i.e., $x = x^* + e$, where e are independently generated from a sub-Gaussian distribution. In this paper, we set $\varepsilon = 0.2$.

Indian Villages. The Indian village datasets consist of 75 rural villages in Karnataka, a state in southern India[2] (Banerjee et al., 2008). These villages are spaced far enough apart from each other that they can be considered independent graphs. We consider 48 villages with ground truth networks, which are also used in Leng et al. (2020). The number of nodes in different network graphs ranges from 77 to 356, and there are a total of 10 actions including the number of rooms, number of beds, the presence of electricity, and other properties. For actions that are categorical categories, we use one-hot encoding, and numerical actions are handled as continuous features.

Foursquare. On Foursquare[3], users can follow their friends' check-in information, including locations and frequency. They may also follow each other's activities and discover new places based on their friends' check-ins and recommendations. We consider two cities: New York and São Paulo. For each city, we first select the category with the most venues and keep all users who have checked in at venues within that category. Then, we calculate the number of their check-ins as actions. Additionally, we remove venues that have never been checked in. To evaluate the algorithm's performance, we use the largest connected component of the social network as the ground truth.

Yelp. We consider two different states: Pennsylvania (PA) and Louisiana (LA) states [4]. For each state, we first select businesses that are located within that state and belong to the food/restaurants category. We then remove users who have not provided reviews or ratings for any businesses from the original dataset. We reconstruct a new graph using the original user-user graph. The largest connected component in this reconstructed graph is then considered as the ground truth. We compute users' ratings and the number of words in reviews as different features in our analysis.

D.4. Datasets

Table 6. Statistics of real-world datasets. For Indian villages, we report the user range across all graphs and other average results.

	#Graph	#User	#Action	#Edges	#Community	Edge density
Indian Villages	48	77 - 356	10	2208.25 ± 750.06	19.15 ± 6.43	0.0504 ± 0.0174
New York	1	449	177	1980	14	0.0098
São Paulo	1	678	430	3874	19	0.0084
Louisiana	1	2065	6094	28514	12	0.0067
Pennsylvania	1	2234	15965	40664	8	0.0081

D.5. Additional experiments and results

End-to-end training vs. Two-stage training. The latent variable C is used to determine the dominant prior under different network games. During end-to-end training, an incorrect estimation of C will result in selecting the opposite prior distribution, which in turn leads to learning an inaccurate network structure. Therefore, we exploit a stage-wise training strategy to estimate the interaction type before learning the network structure. To better illustrate the necessity of the

[2]The village dataset can be downloaded from https://doi.org/10.7910/DVN/U3BIHX

[3]The Foursquare dataset can be downloaded from https://sites.google.com/site/yangdingqi/home/foursquare-dataset?pli=1

[4]The Yelp dataset can be downloaded from https://www.yelp.com/dataset

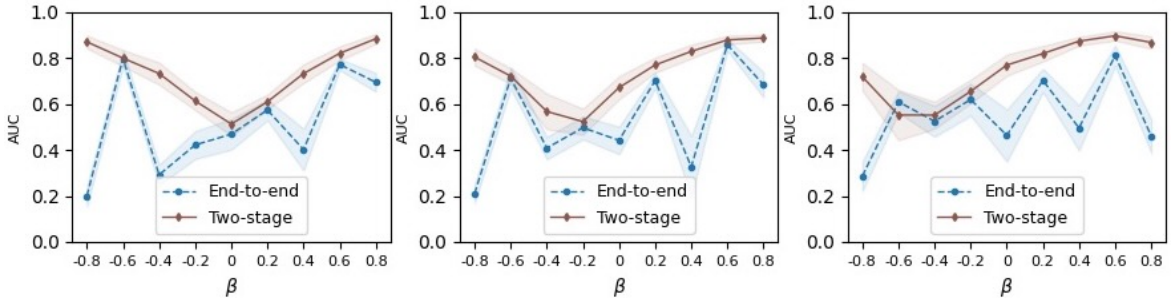

Figure 5. Comparison of end-to-end training and two-stage training on linear quadratic games with different β and α on WS graphs. From left to right: $\alpha = 0.0, 0.5, 0.8$.

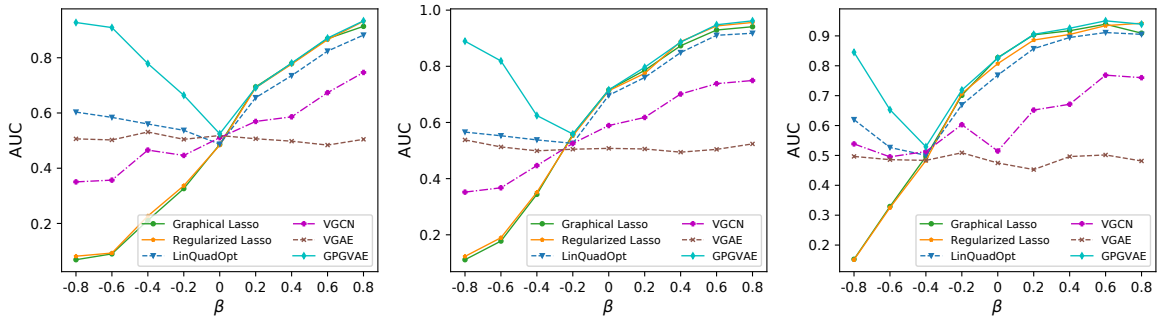

Figure 6. Performance for linear quadratic games with different β and α on ER graphs. From left to right: $\alpha = 0.0, 0.5, 0.8$.

two-stage training strategy, we evaluate the performance of these two learning methods on 10 random graphs with linear quadratic games. Figure 5 shows that the results obtained by the end-to-end training method are not stable under different settings, while the two-stage training framework performs well.

Synthetic. Figures 6 and 7 present the AUC results of all algorithms on linear quadratic games with ER graphs and WS graphs, respectively. GPGVAE outperforms all baselines, especially when there are strategic substitute interactions among individuals, *i.e.*, $\beta < 0$.

Effect of mixture distributions and gradient estimators. Table 7 reports the AUC values obtained on the New York dataset using different gradient estimators and different numbers of mixture distributions. While using a single distribution (*i.e.* $K = 1$) can yield satisfactory results, employing a mixture model can effectively capture dependencies between edges, thereby improving overall results. The two gradient estimators have similar performance when $K > 1$ and the score function with variance reduction technique performs better than the pathwise estimator when using a single distribution.

Table 7. Results with various gradient estimators and different numbers of distributions.

	$K = 1$	$K = 5$	$K = 10$
Pathwise	56.41 ± 0.93	**57.64** ± 0.52	57.59 ± 0.37
Score function*	57.15 ± 0.46	**57.53** ± 0.35	57.14 ± 0.24

Computation complexity. Like most graph learning methods, we need to estimate the probability of each edge, and since we use a mixture of Bernoulli distributions, the computational complexity of GPGVAE method is $\mathcal{O}(N^2K)$, where N is the number of nodes, and K is the number of mixture components. Figure 8 shows the impact of different factors on GPGVAE's training time. Among these factors, the number of nodes has the most significant impact on training time. We also compare the training (second) and inference times (millisecond) of GPGVAE with NodeFormer, where the inference time denotes the time used to infer the network from the node feature based on the trained network structure encoder $q_\phi(A|X)$. The results are reported in Table 8. Although GPGVAE may have longer training times for large datasets, its inference time is shorter.

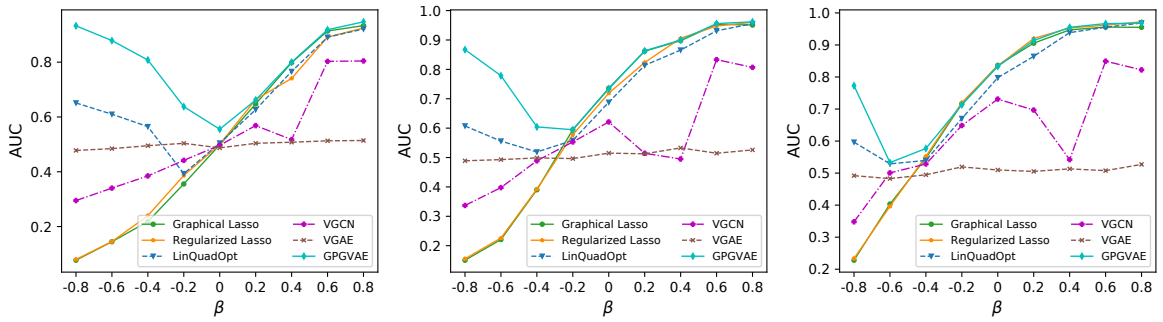

Figure 7. Performance for linear quadratic games with different β and α on WS graphs. From left to right: $\alpha = 0.0, 0.5, 0.8$.

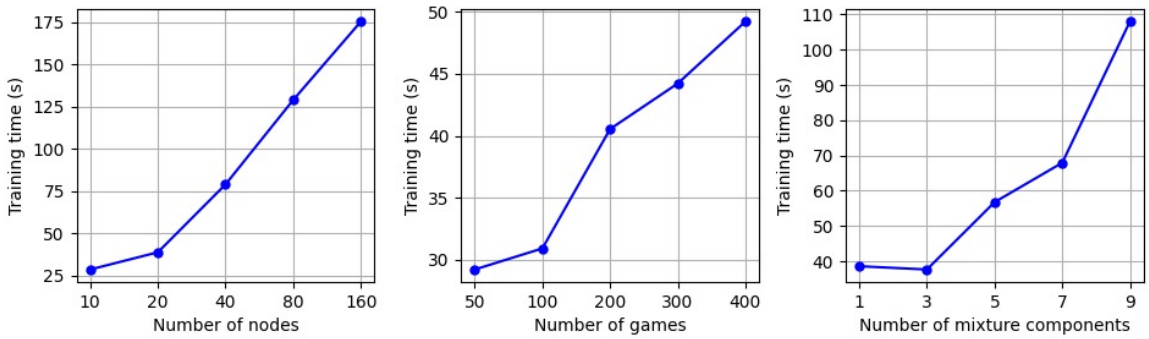

Figure 8. Effect of different factors on the training time of GPGVAE under linear quadratic games.

However, NodeFormer needs to compute the probability for each edge at each layer, leading to longer inference times.

Table 8. Training (second) and inference times (millisecond) for GPGVAE and NodeFormer.

	Method	Linear influence	BH graphical	New York	LA ratings
Training time	NodeFormer	41.27 $\pm$ 6.83e-02	41.47 $\pm$ 8.15e-02	49.44 $\pm$ 2.74e-01	85.55 $\pm$ 7.75e-01
	GPGVAE	38.26 $\pm$ 2.80e-00	35.58 $\pm$ 8.34e-01	273.55 $\pm$ 2.05e-01	1092.24 $\pm$ 7.89e-01
Inference time	NodeFormer	16.91 $\pm$ 1.25e-01	16.93 $\pm$ 2.60e-01	18.10 $\pm$ 4.71e-02	16.42 $\pm$ 4.71e-01
	GPGVAE	3.30 $\pm$ 1.63e-02	3.43 $\pm$ 4.11e-02	3.92 $\pm$ 2.32e-02	4.50 $\pm$ 9.63e-02