A Variational Edge Partition Model for Supervised Graph Representation Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph neural networks (GNNs), which propagate the node features through the edges and learn how to transform the aggregated features under label supervision, have achieved great success in supervised feature extraction for both node-level and graph-level classification tasks. However, GNNs typically treat the graph structure as given and ignore how the edges are formed. This paper introduces a graph generative process to model how the observed edges are generated by aggregating the node interactions over a set of overlapping node communities, each of which contributes to the edges via a logical OR mechanism. Based on this generative model, we partition each edge into the summation of multiple community-specific weighted edges and use them to define community-specific GNNs. A variational inference framework is proposed to jointly learn a GNN-based inference network that partitions the edges into different communities, these community-specific GNNs, and a GNN-based predictor that combines community-specific GNNs for the end classification task. Extensive evaluations on real-world graph datasets have verified the effectiveness of the proposed method in learning discriminative representations for both node-level and graph-level classification tasks.

1 Introduction

Many real-world entities are bonded by relations, *e.g.*, the users in a social network service are connected by online friendships and the atoms in molecules are held together by chemical bonds. Representing the entities by nodes and relations by edges, a set of interconnected entities naturally forms a graph. Reasoning about these entities and their relations could be conducted under graph neural networks (GNNs) [1–4]. Instead of isolating the feature transformation for each individual entity, GNNs allow the information to be exchanged between related entities. Models employing this strategy have achieved great success in a wide range of applications involving graph structured data, such as classification [5, 6], link prediction [7], recommendation [8, 9], (overlapping) community detection [10, 11], and drug discovery [12].

The essence of many graph-analytic tasks could be summarized as supervised graph representation learning, where the information on the graph is usually characterized by the node features, adjacency matrix, and node or graph labels. A supervised machine learning pipeline is often introduced to embed the nodes or graph under label supervision for a specific classification task. For most of the GNN-based methods, the information on nodes and edges are composited through neighborhood aggregation, *i.e.*, updating the features of each node by combining the features of the surrounding nodes. During this process, the graph structure mostly serves as a given source to indicate the nodes' neighborship [13] or provide the weights for aggregation [5]. Treating the graph structure as given overlooks the latent structures that control the formation of the graph, which, however, could provide

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

valuable information for graph representation learning. In other words, failing to consider the latent graph structures may limit the ultimate potential of this line of work.

One type of latent structure at the hinge of the graph structure and node information is node communities. Their connection to the graph can be explained under a latent-community based graph generation process. For example, both the mixed-membership stochastic blockmodel (MMSB) [14] and edge partition model (EPM) [15] explain the formation of edges by node interactions over overlapping latent communities. Let us consider person u in social network $\mathcal{G}$, some of her social connections may be established through interactions with colleagues as a machine learning researcher, some may originate from her interactions with co-members of the same hobby club, and different types of interactions could overlap to strengthen certain social connections between the nodes.

The node community structure also sets the aspects of node information. In the example of social network $\mathcal{G}$ where person u is affiliated with both a research group and a few hobby groups, due to the diverse nature of these different communities, it is likely that she exhibits different characteristics when interacting with co-members of different communities. Namely, the information on u for the research group may be related to her research expertise, and the information on u for the hobby clubs may be related to her hobbies. Therefore, an ideal solution is to learn a different aspect of node properties for each community that it is affiliated with, and represent the overall node information as an aggregation of all community-specific node properties.

Based on this insight, we develop a *variational edge partition model* (VEPM), which is a generative graph representation learning framework. VEPM models how the edges and labels are generated from overlapping latent communities. Instead of hard assigning a node to a single community, VEPM encodes each node into a K-dimensional vector of scores measuring the strength that the node is affiliated with each of the K communities. From these scores, we compute the intensities of pairwise interactions within each community. Given that information, under the Bernoulli-Poisson link, the edges could be modeled by the logical OR of independently generated binary latent edges [15].

The generation of labels includes community-specific node representation learning and aggregation. A key premise of the first part is to detect the hidden structure specified for each community. In our model, it is achieved through edge partition, *i.e.*, decomposing each edge into a summation of link strengths according to the intensities of community-specific interactions. The edge partition step isolates the link strengths accumulated through each community. With the partitioned edges, we learn K separate GNNs to produce community-specific node representations, then compose them into the overall node embeddings to generate the labels. Evaluation shows that the proposed framework can achieve significant performance enhancement on various node and graph classification benchmarks. We summarize our main contributions as follows:

- We introduce VEPM that utilizes the idea of edge partitioning to extract overlapping latent community structures, which are used to not only enrich node attributes with node-community affiliation scores, but also define community-specific node feature aggregations.
- We formalize the training of VEPM in a variational inference framework, which is powered by GNNs in its latent community inference, representation generation, and label prediction. We further develop an effective pretrain-finetune algorithm for training VEPM.
- We analyze how VEPM works and evaluate it over various real-world network datasets. Empirical results show that the proposed VEPM outperforms many previous methods in various node- and graph-level classification tasks, especially with limited labels.

2 Preliminaries and Related Work

Embedding nodes and graphs with GNNs. Given a node-attributed graph $\mathcal{G}$ with N nodes, its information is generally expressed by a design matrix $\mathbf{X} \in \mathbb{R}^{N\times F}$, whose rows represent node features of dimension F, and an adjacency matrix $\mathbf{A}$ of shape $N \times N$. The nonzero values of $\mathbf{A}$ indicate the weights on the corresponding edges. GNNs [1–5, 13, 16] are multi-layer parametric models that maps $(\mathbf{X}, \mathbf{A})$ to the embedding space. The update of the embedding of node u at the tth GNN layer could be summarized as $\mathbf{h}_u^{(t)} = \mathrm{AGG}(f_{\boldsymbol{\theta}}(\mathbf{h}_u^{(t-1)}), \{f_{\boldsymbol{\theta}}(\mathbf{h}_v^{(t-1)})\}_{v\in\mathbb{N}(u)}, \mathbf{A}_{u,:})$ and $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. Here AGG denotes the neighborhood aggregation function, $\mathbb{N}(u)$ denotes the neighbors of u, and $f_{\boldsymbol{\theta}}(\cdot)$ is a transformation function parameterized with $\boldsymbol{\theta}$. In VEPM, we adopt the aggregation function introduced by Kipf and Welling [5], which can be expressed as $\mathbf{h}_u^{(t)} =$

$\sum_{v\in\{u\}\cup\mathbb{N}(u)} \tilde{\mathrm{A}}_{u,v} f_{\boldsymbol{\theta}}(\mathbf{h}_v^{(t-1)})$, where $\tilde{\mathbf{A}}$ is the normalized adjacency matrix of $\mathcal{G}$ augmented with self-loops. When the task requires a graph-level representation, all node embeddings would be summarized into a single embedding vector. This process is called graph pooling [17–19]. In this work, we focus on improving the overall performance from GNN's perspective and adopt a simple graph pooling method proposed by Xu et al. [6]. In the sequel, we unify the expression of models consisting of GNN layers as $\mathrm{GNN}_{\boldsymbol{\theta}}(\mathbf{X}, \mathbf{A})$.

Community-regularized representation learning. Communities are latent groups of nodes in a graph [20]. The simplest way to use community information to help a supervised learning task is to regularize it with a community-detection related task. For instance, Hasanzadeh et al. [21] and Wang et al. [22] jointly train a deep stochastic blockmodel and a classification model; a similar approach could be found in Liu et al. [23] where the graph reconstruction regularizer is replaced by a modularity metric. Either way, the node embeddings are expected to reflect patterns of community structures and maintain informative to the task at the same time. These methods generally treat node-community affiliation as node embeddings and expect them to contain sufficient information for the downstream task. However, the affiliation strengths of nodes to a community may oversimplify the information that such a community can provide for the downstream task. Unlike these methods, VEPM embeds node information on each community with a specific GNN, where the community structures are reflected by the partitioned adjacency matrices. Aggregating node features with these weighted adjacency matrices naturally incorporates community information into task-learning.

Multi-relational data analysis. Leveraging heterogeneous relations has been extensively studied in the literature of mining Knowledge Graphs [24–26]. The data of these models is usually organized by a multigraph, which is related to our perspective of the latent node interactions that generate the observed graph because the interactions in different communities are potentially heterogeneous, and they are permitted to coexist between a pair of nodes. This leads to a shared high-level idea to break down the original heterogeneous graph into homogeneous factors and analyze each factor with a customized model. However, unlike those multigraphs where the edge types are explicitly annotated, the overlapping heterogeneous node interactions are not observable from the data that VEPM aims to handle, escalating the difficulty of our optimization problem. To deal with the latent variables, we formalize VEPM into a generative model and train it via variational inference.

Graph factorization-based models. The existence of multiple hidden structures in graphs is also noticed by some previous work [27, 28]. The architecture of these models starts with graph factorization, followed by a set of modules with each one processing the information from one of the graph factors. In these methods, the only ground-truth information that trains graph factorization is from label supervision, making them rely on excessive label annotations, which might be hard to suffice in practice and thus potentially hinder their application. On account of the potential heterogeneity of graph information on different communities, VEPM adopts a similar pipeline to these methods, but the factorization of the original graph is driven by not only the supervised task but also a graph generative model, which greatly reduces our dependency on the amount of labeled data.

3 Variational Edge Partition Model

Many analytical tasks on node-attributed graphs could boil down to (semi-) supervised classification problems, *i.e.*, given $(\mathbf{X}, \mathbf{A})$ and observed node or graph labels $\mathbf{y}_o$, predicting the unobserved labels $\mathbf{y}_u$. In VEPM, we formalize the classification task as modeling the predictive distribution as $p(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{y}_o)$, and construct our variational supervised learning pipeline with GNNs.

3.1 Generative task-dependent graph representation learning with latent communities

To model $p(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{y}_o)$, we introduce $\boldsymbol{\Phi} \in \mathbb{R}_+^{N\times K}$, a latent non-negative node-community affiliation matrix, whose entry at the nth row and kth column is interpreted as the strength that the nth node is affiliated with the kth community. Given $\boldsymbol{\Phi}$, we specify a *generative network* as

$$\mathbf{A} \sim p_{\boldsymbol{\theta}}(\mathbf{A} \,|\, \boldsymbol{\Phi}), \quad \mathbf{y}_o \sim p_{\boldsymbol{\theta}}(\mathbf{y} \,|\, \mathbf{X}, \mathbf{A}, \boldsymbol{\Phi}) \tag{1}$$

to describe how the observed edges $\mathbf{A}$ and labels $\mathbf{y}_o$ are generated. Due to the complexity brought by the deep architecture of the *generative network*, analytically inferring the posterior of $\boldsymbol{\Phi}$ is impractical. We hence approximate the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{\Phi} \,|\, \mathbf{A}, \mathbf{y}_o)$ with a variational distribution, $q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \,|\, \mathbf{A}, \mathbf{X})$, modeled by a separate GNN-based *inference network*. The subscripts $_{\boldsymbol{\theta}}$ and $_{\boldsymbol{\omega}}$ denote the parameters

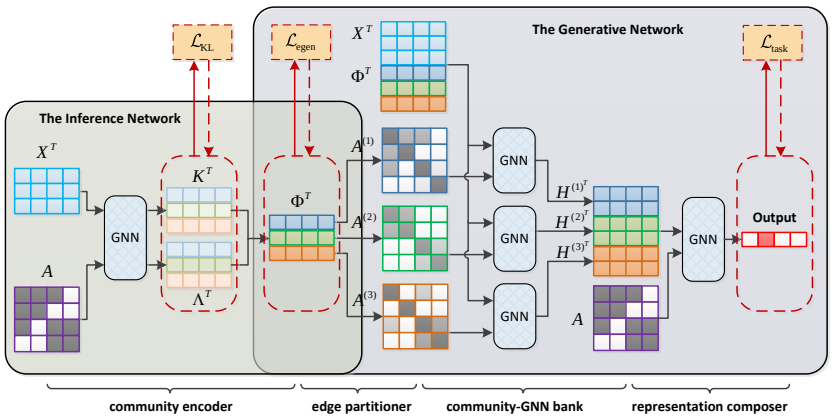


Figure 1: The overview of VEPM's computation graph. The input data successively passes four modules described in Sections 3.2 and 3.3: the *community encoder*, *edge partitioner*, *community-GNN bank*, and *representation composer*. Node information is processed community-wise by the *edge partitioner* and the *community-GNN bank*. The former factorizes the original graph into a set of weighted graphs by the relative inter-node interaction rates across communities; the latter performs neighborhood aggregation with the factorized graphs, such that in each community, nodes exchange more information with intensively interacted neighbors. Hence the obtained node representations are community-specific. Please note that we use the transposed notation for $\mathbf{\Phi}$ and node embeddings in this diagram, so the concatenations are along the vertical direction. Solid and dashed arrows indicate network forwarding and gradient back-propagation, respectively.

in the *generative network* and *inference network*, respectively. We illustrate the architecture of VEPM in Figure 1. We approximate the posterior predictive distribution using the Monte Carlo method as

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{y}_o) \approx \tfrac{1}{S} \textstyle\sum_{s=1}^{S} p_{\boldsymbol{\theta}}(\mathbf{y}_u \,|\, \mathbf{X}, \mathbf{A}, \mathbf{\Phi}^{(s)}), \tag{2}$$

where $\mathbf{\Phi}^{(s)} \overset{iid}{\sim} q_{\boldsymbol{\omega}}(\mathbf{\Phi} \,|\, \mathbf{A}, \mathbf{X})$ for $s = 1, \ldots, S$.

In what follows, we outline the data generation process and corresponding modules in the *generative network*, introduce the module in the *inference network*, and describe how to train both networks.

3.2 Edge generation and label prediction

To model the distribution of the edges given $\mathbf{\Phi}$, we adopt the generative process developed in EPM [15], which explains the generation of the edges under the Bernoulli-Poisson link as $\mathrm{A}_{i,j} = \mathbf{1}_{\mathrm{M}_{i,j} \geq 1}$, $\mathrm{M}_{i,j} \sim \mathrm{Poisson}\left(\sum_{k=1}^{K} \gamma_k \Phi_{i,k} \Phi_{j,k}\right)$. Here γ_k is a positive community activation level indicator, which measures the member interaction frequency via community k, and in our practice is treated as a trainable parameter shared by both the *inference network* and *generative network*. We interpret $\gamma_k \Phi_{i,k} \Phi_{j,k}$ as the interaction rate between nodes i and j via community k.

The EPM has an alternative representation that partitions each edge into the logical disjunction ($i.e.$, logical OR) of K latent binary edges, expressed as

$$\mathrm{A}_{i,j} = \vee_{k=1}^{K} \mathrm{A}_{i,j,k}, \; \mathrm{A}_{i,j,k} \sim \mathrm{Ber}(p_{ijk}),$$

where $p_{ijk} := 1 - e^{-\gamma_k \Phi_{i,k} \Phi_{j,k}}$. Thus nodes i and j would be connected as long as their interaction forms up an edge in at least one community. In other words, they are disconnected only if their interactions in all communities fail to generate any edges. Under the EPM, we have the conditional probability as $p_{\boldsymbol{\theta}}(\mathrm{A}_{i,j} = 1 \,|\, \mathbf{\Phi}) = 1 - e^{-\sum_{k=1}^{K} \gamma_k \Phi_{i,k} \Phi_{j,k}}$. To complete the model, we set the prior distribution of the node-community affiliation matrix as $\mathbf{\Phi} \sim \prod_{i=1}^{N} \prod_{k=1}^{K} \mathrm{Gam}(\Phi_{i,k}; \alpha, \beta)$.

Given $\mathbf{A}$ and $\mathbf{\Phi}$, we now describe the predictive process of the labels. The node-community affiliation impacts this process from two aspects: for each node, the corresponding row of $\mathbf{\Phi}$ serves as side information that could enrich node attributes; for each pair of connected nodes, it could be used to derive the node interactions rate via each community, *i.e.*, $\{\gamma_k \Phi_{i,k} \Phi_{j,k}\}_{k=1,K}$, which are further used to partition the edges to K communities, carried out by the *edge partitioner*:

Edge partitioner. *The edge partitioner takes edges* $\mathbf{A}$ *and node-community affiliation matrix* $\mathbf{\Phi}$ *as inputs and returns* K *positive-weighted edges:* $\{\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(K)}\}$, s.t. $\sum_{k=1}^{K} \mathbf{A}^{(k)} = \mathbf{A}$. *The*

edge partition function is

$$\mathrm{A}_{i,j}^{(k)} = \mathrm{A}_{i,j} \cdot \frac{e^{(\gamma_k \Phi_{i,k} \Phi_{j,k})/\tau}}{\sum_{k'} e^{(\gamma_{k'} \Phi_{i,k'} \Phi_{j,k'})/\tau}},\ k \in \{1, \ldots, K\},\ i, j \in \{1, \ldots, N\}, \tag{3}$$

where τ is a "temperature" that controls the sharpness of partition.

The values of τ are specified in Appendix C. The effect of the *edge partitioner* could be considered as a soft assignment of each edge into different communities. Setting temperature τ to be low could drive the soft assignment towards a hard assignment, and consequently, when aggregating node features with $\{\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(K)}\}$, the edge partitioner would concentrate the information exchange between any two connected nodes at one community. In other words, the mutual influence between two nodes, measured by the edge weight, is relatively high at the community that contributes the most of the interactions between the pair, while being relatively weak at the other communities.

In our implementation, we further generalize Equation (3) to a metacommunity-based edge partition, specifically, by replacing $\gamma_k \Phi_{i,k} \Phi_{j,k}$ with $\mathbf{\Phi}_{i,k} \mathrm{diag}(\boldsymbol{\gamma}_k) \mathbf{\Phi}'_{j,k}$. In the new expression, $\mathbf{\Phi}_{i,k}$ denotes the kth segment in node i's community-affiliation encoding, $\boldsymbol{\gamma}_k$ is a vector of the activation levels for communities in the kth metacommunity. This generalization allows the total number of communities to be greater than K, enhancing the model implementation flexibility at a minor interpretability cost.

The *edge partitioner* provides a soft separation of the neighborhood aggregation routes for each community, which is passed to the *community-GNN bank*:

Community-GNN bank. *The module takes $\mathbf{X}^* := \mathbf{X} \,\|\, \mathbf{\Phi}$ as input, where the operator $\|$ denotes "concatenation," and learns community-specific node embeddings with K separate GNNs, i.e., the outputs are $\mathrm{g}_{\boldsymbol{\theta}}^{(1)}(\mathbf{X}^*),\ \mathrm{g}_{\boldsymbol{\theta}}^{(2)}(\mathbf{X}^*),\ \cdots,\ \mathrm{g}_{\boldsymbol{\theta}}^{(K)}(\mathbf{X}^*)$, where $\mathrm{g}_{\boldsymbol{\theta}}^{(k)}(\cdot) := \mathrm{GNN}_{\boldsymbol{\theta}}\big(\cdot, \mathbf{A}^{(k)}\big),\ k \in \{1, \ldots, K\}$.*

The intention of the *edge partitioner* and *community-GNN bank* is to capture the node information specific to each community. For instance, in a social network, such kind of information could be the different social roles of people when considered affiliated with different social groups, *e.g.*, one may be a former computer science major student in an alumni group, a research scientist in a work group, and an amateur chess player in a hobby club group. As shown in Section 4.3, the node embeddings produced by the *community-GNN bank* are separable by communities.

We finally introduce the *representation composer*, which embeds a combination of the information provided by each community into a global representation via a GNN:

Representation composer. *Let $\mathbf{H}^{(k)} := \mathrm{g}_{\boldsymbol{\theta}}^{(k)}(\mathbf{X}^*)$ denote the node representations learned from the kth community, where $k = 1, \ldots, K$, and $f(\cdot)$ denote the representation composer, whose functionality is to project a composite of community-specific node representations to one representation matrix, i.e., $\mathbf{H}_{\mathcal{V}} = f\left(\mathbf{H}^{(1)}, \mathbf{H}^{(2)}, \cdots, \mathbf{H}^{(K)}\right) := \mathrm{GNN}_{\boldsymbol{\theta}}\left(\|_{k=1}^{K} \mathbf{H}^{(k)}, \mathbf{A}\right)$.*

For graph-level tasks, we further pool node embeddings into a single vector representation $\mathbf{h}_{\mathcal{G}}$, as in Xu et al. [6]. Taking `softmax` on the feature dimension of $\mathbf{H}_{\mathcal{V}}$ or $\mathbf{h}_{\mathcal{G}}$ gives the predicted probabilities of labels, from which we are able classify the unlabeled objects. Cascading the *edge partitioner* and *community-GNN bank* with the *representation composer* yields the probability $p_{\boldsymbol{\theta}}(\mathbf{y} \,|\, \mathbf{X}, \mathbf{A}, \mathbf{\Phi})$.

3.3 Variational latent community inference

We use a *community encoder* as the module in the *inference network* to approximate the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{\Phi} \,|\, \mathbf{A}, \mathbf{y})$, and provide the *generative network* with the critical latent variable $\mathbf{\Phi}$:

Community encoder. *The community encoder models the variational posterior $q_{\boldsymbol{\omega}}(\mathbf{\Phi} \,|\, \mathbf{A}, \mathbf{X})$ by a Weibull distribution with shape $\mathbf{K}$ and scale $\mathbf{\Lambda}$, whose parameters are learned by a GNN as $\mathbf{K} \,\|\, \mathbf{\Lambda} = \mathrm{GNN}_{\boldsymbol{\omega}}\big(\mathbf{X}, \mathbf{A}\big)$. A Weibull random sample from $q_{\boldsymbol{\omega}}(\mathbf{\Phi} \,|\, \mathbf{A}, \mathbf{X})$ could be created through the inverse CDF transformation of a uniform random variable, given as follows:*

$$\mathbf{\Phi} = \mathbf{\Lambda} \odot \big(-\log(1 - \mathbf{U})\big)^{(1 \oslash \mathbf{K})}, \mathrm{U}_{i,k} \overset{iid}{\sim} \mathrm{U}(0, 1),\ \ \forall (i, k) \in \{1, \ldots, N\} \times \{1, \ldots, K\}, \tag{4}$$

where $\odot$ and $\oslash$ denote element-wise multiplication and division.

3.4 The overall training algorithm

We train VEPM by optimizing the evidence lower bound (ELBO), decomposed into three terms, as

$$\mathcal{L} = \mathcal{L}_{\mathrm{task}} + \mathcal{L}_{\mathrm{egen}} + \mathcal{L}_{\mathrm{KL}}, \tag{5}$$

where $\mathcal{L}_{\text{task}} = \mathbb{E}_{q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X})} \log p_{\boldsymbol{\theta}}(\mathbf{y}_o \mid \mathbf{A}, \boldsymbol{\Phi}, \mathbf{X})$, $\mathcal{L}_{\text{egen}} = \mathbb{E}_{q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X})} \log p_{\boldsymbol{\theta}}(\mathbf{A} \mid \boldsymbol{\Phi})$, and $\mathcal{L}_{\text{KL}} = -D_{\text{KL}}\big(q_{\boldsymbol{\omega}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{X}) \,\|\, p(\boldsymbol{\Phi})\big)$. These three terms correspond to the classification task, edge generation, and KL-regularization, respectively. Note that our specifications of $\boldsymbol{\Phi}$'s prior and variational posterior yield an analytical expression of $\mathcal{L}_{\text{KL}}$, as described in detail in Appendix B.

Since modeling the likelihood of the labels involves stacking multiple modules embodied by deep neural networks, approximating the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{\Phi} \mid \mathbf{A}, \mathbf{y}_o)$ is difficult from an optimization perspective. To effectively train VEPM, we leverage the edge generation process and break down the overall training pipeline into two phases: an *unsupervised pretrain*, followed by a *supervised finetune*. Specifically, we first pretrain the *inference network* by optimizing $\mathcal{L}_{\text{egen}} + \mathcal{L}_{\text{KL}}$ to link latent communities with edge generation. We then join the pretrained *inference network* up with the *generative network*, and train the entire architecture with the objective function given in Equation (5). In this stage, the latent communities are finetuned by $\mathcal{L}_{\text{task}}$ to align them to the task. The efficacy of pretraining the model with an unsupervised loss is also expressed in other work [29] that involves optimizing graph structure for GNNs with a graph autoencoder. We summarize the overall training pipeline in Appendix A, and empirically study the effect of bi-phase training algorithm in Section 4.2.

4 Empirical Evaluation

4.1 Node & graph classification

Datasets & experimental settings. For node classification, we consider three citation networks: Cora, Citeseer, and Pubmed, which provide bag-of-words document representations as node features and (undirected) citations as edges, and use the train/test/validation split standardized by Kipf and Welling [5]; For graph classification, we consider four bioinformatics datasets (MUTAG, PTC, NCI1, PROTEINS) and four social network datasets (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY and REDDIT-MULTI). The input node features are crafted in the same way as Xu et al. [6]. Most of the baselines are compared following the 10-fold cross validation based evaluation protocol proposed by Xu et al. [6]. Since this protocol is criticized for being prone to overestimating model performance [30], we also evaluate our model following Zhang and Chen [31], who conduct a more rigorous train-validation-test protocol. To counter the memory complexity expansion by using K GNNs in the *community-GNN bank*, we limit the size of hidden dimension in each *community-GNN* to $1/K$ of what is commonly used among the baselines (*e.g.*, 64 for both tasks). We elaborate other details of the benchmarks, model architecture, and hyperparameter determination in Appendix C.

Node classification. We use classification accuracy as the evaluation metric for node classification. Table 1 reports the average performance of VEPM ($\pm$ standard error) against related baselines that are categorized into three different groups. The first group consists of the GCNs [5] and its variant GCN-64 that expands the hidden dimension from 16 to 64. VEPM outperforms the first group by a significant margin. The explanation is twofold: (i) VEPM augments the node attributes with community information carried by the inferred node-community affiliations, (ii) VEPM learns community-specific node embeddings. The second group includes SIG-VAE [21] and WGCAE [22], both of which learn node embeddings from graph generative models jointly optimized with a supervised loss. The performance gain that VEPM obtains could be attributed to our unique label predictive process, which not only appends extra community information to node attributes, but also injects structural patterns of communities into neighborhood aggregation. The third group [13, 32, 27] is focused on learning node embeddings leveraging heterogeneous hidden relations. When fitting the hidden relations via attention mechanism, they only use information from the node features through label supervision, whereas our approach also takes the observed graph structure into account via a graph generative model. The additional information from the graph utilized by VEPM could explain the enhancement achieved by VEPM over the third group.

Table 1: Comparison of node classification performance. The asterisk ($*$) marks the results reported by Veličković et al. [13].

Method	Cora	Citeseer	Pubmed
ChebNet [4]	81.2	69.8	74.4
GCN [5]	81.5	70.3	79.0
GCN-64*	81.4	70.9	79.0
SIG-VAE [21]	79.7	70.4	79.3
WGCAE [22]	82.0	72.1	79.1
GAT* [13]	83.0	72.5	79.0
hGANet [32]	83.5	72.7	79.2
DisenGCN [27]	83.7	**73.4**	80.5
VEPM (this work)	**84.3** $\pm$ 0.1	72.5 $\pm$ 0.1	**82.4** $\pm$ 0.2

Graph classification. For the first protocol, we compare VEPM with classical graph classification baselines [33–35], generic-GNN-based models [13, 32, 6], and GNNs with relation-based or task-

driven graph factorization [25, 26, 28]. Results in Table 2 show that VEPM achieves the best graph classification performance on 5 out of 8 benchmarks, and the second-best performance on the other 3 benchmarks, including NCI1, where VEPM outperforms all the other GNN-based models.

Table 2: Comparison of graph classification performance (average accuracy $\pm$ standard error).

Method	IMDB-B	IMDB-M	MUTAG	PTC	NCI1	PROTEINS	RDT-B	RDT-M
WL subtree [33]	73.8 $\pm$ 3.9	50.9 $\pm$ 3.8	90.4 $\pm$ 5.7	59.9 $\pm$ 4.3	**86.0** $\pm$ 1.8	75.0 $\pm$ 3.1	81.0 $\pm$ 3.1	52.5 $\pm$ 2.1
PATCHYSAN [34]	71.0 $\pm$ 2.2	45.2 $\pm$ 2.8	92.6 $\pm$ 4.2	60.0 $\pm$ 4.8	78.6 $\pm$ 1.9	75.9 $\pm$ 2.8	86.3 $\pm$ 1.6	49.1 $\pm$ 0.7
AWE [35]	74.5 $\pm$ 5.9	51.5 $\pm$ 3.6	87.9 $\pm$ 9.8	-	-	-	87.9 $\pm$ 2.5	54.7 $\pm$ 2.9
GAT [13]	70.5 $\pm$ 2.3	47.8 $\pm$ 3.1	89.4 $\pm$ 6.1	66.7 $\pm$ 5.1	-	70.5$\pm$3.6	-	-
hGANet [32]	-	49.0	90.0	65.0	-	78.7	-	-
GIN [6]	75.1 $\pm$ 5.1	52.3 $\pm$ 2.8	89.4 $\pm$ 5.6	64.6 $\pm$ 7.0	82.7 $\pm$ 1.6	76.2 $\pm$ 2.8	**92.4** $\pm$ 2.5	**57.5** $\pm$ 1.5
R-GCN [25]	-	-	82.3 $\pm$ 9.2	67.8 $\pm$ 13.2	-	-	-	-
CompGCN [26]	-	-	89.0 $\pm$ 11.1	71.6 $\pm$ 12.0	-	-	-	-
FactorGCN [28]	75.3 $\pm$ 2.7	51.0 $\pm$ 3.1	89.9 $\pm$ 6.5	74.3 $\pm$ 8.9	76.4 $\pm$ 2.1	74.2$\pm$4.6	-	-
VEPM (this work)	**76.7** $\pm$ 3.1	**54.1** $\pm$ 2.1	**93.6** $\pm$ 3.4	**75.6** $\pm$ 5.9	83.9 $\pm$ 1.8	**80.5** $\pm$ 2.8	90.5 $\pm$ 1.8	55.0 $\pm$ 1.5

For the second protocol, aside from a traditional method, WL [33], and a generic-GNN-based method, DGCNN [36], we compare VEPM with two GNNs [37, 31] that adapt capsule neural networks [38] to graph-structured data, which aim to learn different aspects of graph properties via dynamic routing [39]. The results in Table 3 show that VEPM outperforms these GNN-based methods on most of the benchmarks. This indicates that the aspects of graph information, as defined with communities and learned via aggregating node features with partitioned graphs, are more pertinent to the end tasks.

Table 3: Graph classification accuracies under the protocol by Zhang and Chen [31].

Method	IMDB-B	IMDB-M	MUTAG	NCI1	PROTEINS	RDT-M
WL subtree [33]	73.4 $\pm$4.6	49.3 $\pm$4.8	82.1 $\pm$0.4	82.2 $\pm$0.2	74.7 $\pm$0.5	49.4 $\pm$2.4
DGCNN [36]	70.0 $\pm$0.9	47.8 $\pm$0.9	85.8 $\pm$1.7	74.4 $\pm$0.5	75.5 $\pm$0.9	48.7 $\pm$4.5
GCAPS-CNN [37]	71.7 $\pm$3.4	48.5 $\pm$4.1	-	**82.7** $\pm$2.4	76.4 $\pm$4.2	50.1 $\pm$1.7
CapsGNN [31]	73.1 $\pm$4.8	50.3 $\pm$2.7	86.7 $\pm$6.9	78.4 $\pm$1.6	76.3 $\pm$3.6	**52.9** $\pm$1.5
VEPM (this work)	**74.6**$\pm$2.1	**50.6** $\pm$3.6	**91.5** $\pm$ 4.3	81.7 $\pm$ 1.4	**76.8** $\pm$ 4.1	50.2 $\pm$ 2.1

Classification with reduced labels. The gain of incorporating the graph generation process into VEPM is that the observed graph structure also provides information for hidden community detection and edge partition, which is beneficial if the amount of labeled data is not sufficient for both decomposing the graph into hidden factors and semi-supervised task-learning. To illustrate this point, based on the evaluation protocol by Kipf and Welling [5] for node classification and Xu et al. [6] for graph classification, we re-evaluate the performance of VEPM under reduced training labels on Cora (for node classification) and MUTAG (for graph classification), along with generic-GNN-based methods [5, 13, 6] and decomposition-based methods [27, 28]. The results are shown in Figure 2: the performance of all models is negatively impacted by the reduction in training labels, those suffering from the strongest performance decline are DisenGCN [27] and FactorGCN [28] that decompose graphs solely based on label supervision, while VEPM also performs graph factorization with limited labels, it consistently outperforms the other methods under all settings, and the superiority becomes even more evident as the amount of annotated data further decreases. We consider the enhanced robustness under sparsely labeled data as a crucial feature to have for real-world graphs.

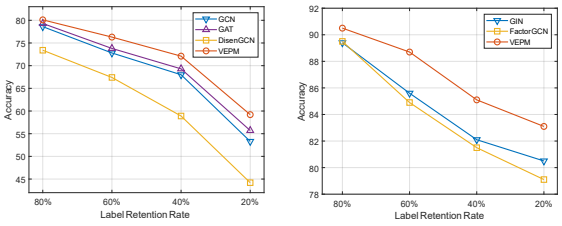


Figure 2: Node classification accuracy on Cora (left) and graph classification accuracy on MUTAG (right) as the number of labels decreases. The horizontal axis is the keep rate of labels in the training set under corresponding protocols.

4.2 Ablation studies

Various input features. We evaluate the model performance with four types of inputs: random noise, hand-crafted features by Xu et al. [6], node-community affiliation scores (*i.e.*, $\mathbf{\Phi}$), and the combination of the last two. The quality of these inputs are evaluated by the graph classification accuracy as summarized in Table 4. Two conclusions could be drawn: (i) $\mathbf{\Phi}$ is comparably informative to the end task as hand-crafted node features, both are significantly better than random noise (see

rows 1 – 3 of Table 4); (ii) the information $\mathbf{\Phi}$ and $\mathbf{X}$ hold for the task enhances the discriminative level of learned representations in a complementary manner (see rows 2, 4 of Table 4). These results verify the quality of $\mathbf{\Phi}$ inferred by VEPM as additional node features.

Table 4: Comparison of VEPMs with various input node features.

Random	Hand-crafted	Community-based	IMDB-B	IMDB-M	MUTAG	PTC	NCI1	PROTEINS
✔			64.7 ± 1.6	42.3 ± 1.5	84.6 ± 4.3	63.6 ± 2.0	67.5 ± 2.1	76.5 ± 3.5
	✔		$\mathbf{80.3} \pm 2.0$	53.5 ± 2.6	93.1 ± 5.0	74.7 ± 4.1	68.0 ± 2.1	77.9 ± 2.7
		✔	74.7 ± 5.1	51.5 ± 2.0	84.6 ± 5.5	68.9 ± 3.9	80.8 ± 1.4	75.7 ± 3.8
	✔	✔	76.7 ± 3.1	$\mathbf{54.1} \pm 2.1$	$\mathbf{93.6} \pm 3.5$	$\mathbf{75.6} \pm 5.9$	$\mathbf{83.9} \pm 1.8$	$\mathbf{80.5} \pm 2.8$

With or without pretraining. Finally, we compare the performance of VEPM that undergoes both pretraining and finetuning against that of a VEPM trained with Equation (5) from scratch. Results in Table 5 shows that comparing with training from scratch, the state of the *community encoder* after pretraining is better for the classification tasks, suggesting the benefits brought by pretraining.

Table 5: Comparison of VEPMs with different training schemes.

	IMDB-B	IMDB-M	MUTAG	PTC	NCI1	PROTEINS
trained from scratch	61.0 ± 3.6	46.8 ± 0.8	86.6 ± 6.6	68.6 ± 2.0	67.2 ± 8.4	60.6 ± 4.1
pretrain + finetune	$\mathbf{76.7} \pm 3.1$	$\mathbf{54.1} \pm 2.1$	$\mathbf{93.6} \pm 3.5$	$\mathbf{75.6} \pm 5.9$	$\mathbf{83.9} \pm 1.8$	$\mathbf{80.5} \pm 2.8$

4.3 Qualitative analysis

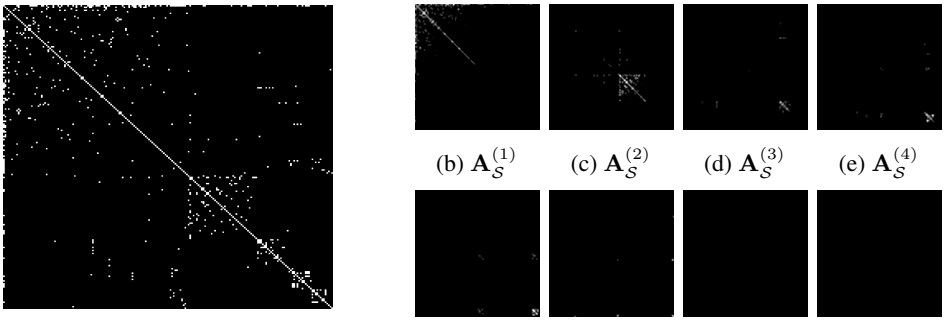
Visualizing community structures. To show that through partitioning the edges, VEPM identifies each latent community from the original graph, in Figure 3, we plot the adjacency matrices of a 200-node subgraph of Cora, before and after edge partition. The subgraph is created via breadth-first search [40] node selection to ensure connectivity. We sort the nodes sample $\mathbb{S}$ in order to present a clearer view of the community structures. Specifically, we prepare K buckets; for node u, we compute the total interactions it engages under metacommunity k by $\mu_{u,k} := \sum_{v \in \mathbb{S}} \mathbf{\Phi}_{u,k} \mathrm{diag}(\boldsymbol{\gamma}_k) \mathbf{\Phi}'_{v,k}$, and assign it to bucket k', where $k' = \arg\max_k \mu_{u,k}$. We first sort the buckets by descending their counts of assigned nodes, then sort the nodes within bucket $k,\ k \in \{1, \ldots, K\}$, by descending value of $\mu_{u,k}$. The $\mathbf{\Phi}$ used for sorting is sampled at the end of training.

The community structures that VEPM detects could be found in Figure 3a as the blocks on the main diagonal. The fact that most of the bright spots (*i.e.*, edges) are located in these on-diagonal blocks reflects dense node connections internal to each community. Sparsely distributed off-diagonal spots provide evidence of overlapping membership between the communities, which is also taken into account by our graph generative model. Figures 3b to 3i show that the *edge partitioner* has done a sound job in separating latent communities. Due to the limited size of the subgraph, some small metacommunities may not contain any large-weighted edges between the nodes in this subgraph, which accounts for the appeared emptiness in Figures 3h and 3i. Such kind of artifact disappears when we replace the subgraph by the full graph of Cora, as visualized in Figures 6 to 14 in Appendix I.

Visualizing latent representations. The previous visualization experiment verifies the following propositions: (i) the identified hidden structures are latent communities, and (ii) the partitioned graphs are different from each other. We now study how they benefit representation learning. Cora and MUTAG are selected as the representatives for node and graph classification benchmarks. For the MUTAG dataset, we remove graphs that contain node categories with less than 5 instances (less than 10 from a totality of 188) and randomly sample 10 graphs for the visualization experiments.

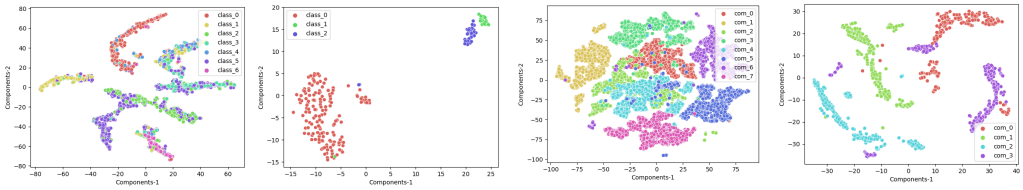
We first visualize $\mathbf{\Phi}$ obtained at the end of the *unsupervised pretrain* stage (Figures 4a and 4b), where the node-community affiliation vectors are projected to 2-D space via t-SNE [41]. Proposition (i) ensures that the node information carried by $\mathbf{\Phi}$ is about communities. We color code the scatters by node labels, both Figures 4a and 4b exhibit strong correlation between the spatial clusters and colors, which indicates that even without label supervision, the node information provided with $\mathbf{\Phi}$ has discriminative power on classifying nodes, which is coherent with the conclusions in Section 4.2.

We then visualize the community-specific node emebddings obtained at the end of the *supervised finetune* stage (Figures 4c and 4d). Similarly, t-SNE is adopted to reduce the dimensionality of obtained node embeddings. This time we color code the scatters by the latent metacommunity they correspond to. Both Figures 4c and 4d exhibit clear boundaries between the colored clusters, which shows that the model is able to extract different information from multiple communities, which



(a) $\mathbf{A}_{\mathcal{S}}$ (b) $\mathbf{A}_{\mathcal{S}}^{(1)}$ (c) $\mathbf{A}_{\mathcal{S}}^{(2)}$ (d) $\mathbf{A}_{\mathcal{S}}^{(3)}$ (e) $\mathbf{A}_{\mathcal{S}}^{(4)}$ (f) $\mathbf{A}_{\mathcal{S}}^{(5)}$ (g) $\mathbf{A}_{\mathcal{S}}^{(6)}$ (h) $\mathbf{A}_{\mathcal{S}}^{(7)}$ (i) $\mathbf{A}_{\mathcal{S}}^{(8)}$

Figure 3: S is a connected subgraph of 200 nodes sampled from Cora with adjacency matrix $\mathbf{A}_{\mathcal{S}}$ as visualized in (a). (b)–(i) are the K outputs from the *edge partitioner*. Brighter color represents larger edge weights. For a clearer visualization effect, we enlarge the bright spots in (b)–(i) by 9 times.



(a) $\mathbf{\Phi}$, Cora (b) $\mathbf{\Phi}$, MUTAG (c) $\mathrm{g}_{\boldsymbol{\theta}}^{(\cdot)}(\mathbf{X}^*)$, Cora (d) $\mathrm{g}_{\boldsymbol{\theta}}^{(\cdot)}(\mathbf{X}^*)$, MUTAG

Figure 4: t-SNE visualization of $\mathbf{\Phi}$ obtained from *unsupervised pretrain* on (a) Cora and (b) MUTAG, as well as community-specific node embeddings obtained during *supervised finetune* on (c) Cora and (d) MUTAG.

enhances the overall expressiveness of learned node or graph representations and potentially leads to better model performance on the downstream tasks.

4.4 Complexity analysis

Recall that N, M, K denote the numbers of nodes, edges, and metacommunities in the graph; F denotes the feature dimension; and L_1, L_2, L_3 denote the number of layers in the *community encoder*, in the *community-GNN bank*, and in the *representation composer*. The time complexity of training VEPM is $\mathcal{O}((L_1 + L_2 + L_3)MF + (L_1 + {}^{L_2}/_{K} + L_3)NF^2 + N^2F)$. For a sparse graph where $N^2 \gg M$, the computational overhead can be reduced to $\mathcal{O}(M)$ [1] if graph reconstruction is accelerated via subsampling the nodes to $\mathcal{O}(\sqrt{M})$, as in Salha et al. [42]. Effects and implications of adopting such type of acceleration algorithms are discussed in Appendix E. The space complexity of VEPM is $\mathcal{O}((L_1+L_2+L_3)NF+KM+(L_1+{}^{L_2}/_{K}+L_3)F^2)$, among which $\mathcal{O}((L_1+{}^{L_2}/_{K}+L_3)F^2)$ is contributed by model parameters. It is noteworthy to point out that the memory cost of graph reconstruction is manageable by computing the dense matrix multiplication block-wise with a fixed maximum block size. From all aspects of complexity, VEPM (with acceleration) is comparable to GATs [13, 32] and models involving graph factorization [27, 28].

5 Conclusion

Moving beyond treating the graph adjacency matrix as given, we develop variational edge partition models (VEPMs) to extract overlapping node communities and perform community-specific node feature aggregations. Specifically, we first utilize a GNN-based inference network to obtain node-community affiliation strengths, with which we augment node attributes, and partition the edges according to the intensities of node interactions with respect to each community. We learn GNN-based node embeddings for each community by aggregating node features with the corresponding partitioned graph, and aggregate all community-specific node embeddings for the downstream tasks. Extensive qualitative and quantitative experiments on both node-level and graph-level classification tasks are performed to illustrate the working mechanism and demonstrate the efficacy of VEPMs in supervised graph representation learning.

[1] For expression simplicity, the scale of F is treated as constant, which is ubiquitous in practice.

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[2] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[3] James Atwood and Don Towsley. Diffusion-convolutional neural networks. volume 29, 2016.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[7] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[8] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[9] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. LightGCN: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[10] Fan-Yun Sun, Meng Qu, Jordan Hoffmann, Chin-Wei Huang, and Jian Tang. vGraph: A generative model for joint community detection and node representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[11] Nikhil Mehta, Lawrence Carin, and Piyush Rai. Stochastic blockmodels meet graph neural networks. In *International Conference on Machine Learning (ICML)*, pages 4466–4474, 2019.

[12] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. GraphAF: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations (ICLR)*, 2020.

[13] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[14] Edo M Airoldi, David Blei, Stephen Fienberg, and Eric Xing. Mixed membership stochastic blockmodels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 21, 2009.

[15] Mingyuan Zhou. Infinite edge partition models for overlapping community detection and link prediction. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 38 of *Proceedings of Machine Learning Research*, pages 1135–1143, 2015.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[17] Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *International Conference on Learning Representations (ICLR)*, 2020.

[18] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *International Conference on Learning Representations (ICLR)*, 2020.

[19] Amirhossein Nouranizadeh, Mohammadjavad Matinkia, Mohammad Rahmati, and Reza Safabakhsh. Maximum entropy weighted independent set pooling for graph neural networks. *arXiv preprint arXiv:2107.01410*, 2021.

[20] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 695–704, 2008.

[21] Arman Hasanzadeh, Ehsan Hajiramezanali, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Semi-implicit graph variational auto-encoders. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 10711–10722, 2019.

[22] Chaojie Wang, Hao Zhang, Bo Chen, Dongsheng Wang, Zhengjue Wang, and Mingyuan Zhou. Deep relational topic modeling via graph Poisson gamma belief network. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

[23] Yanbei Liu, Qi Wang, Xiao Wang, Fang Zhang, Lei Geng, Jun Wu, and Zhitao Xiao. Community enhanced graph convolutional networks. *Pattern Recognition Letters*, 138:462–468, 2020.

[24] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[25] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference (ESWC)*, pages 593–607. Springer, 2018.

[26] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[27] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. Disentangled graph convolutional networks. In *International Conference on Machine Learning (ICML)*, pages 4212–4221, 2019.

[28] Yiding Yang, Zunlei Feng, Mingli Song, and Xinchao Wang. Factorizable graph convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 20286–20296, 2020.

[29] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11015–11023, 2021.

[30] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations (ICLR)*, 2020.

[31] Xinyi Zhang and Lihui Chen. Capsule graph neural network. In *International Conference on Learning Representations (ICLR)*, 2019.

[32] Hongyang Gao and Shuiwang Ji. Graph representation learning via hard and channel-wise attention networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 741–749, 2019.

[33] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[34] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning (ICML)*, pages 2014–2023, 2016.

[35] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International conference on machine learning (ICML)*, pages 2186–2195, 2018.

[36] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[37] Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *arXiv preprint arXiv:1805.08090*, 2018.

[38] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[39] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[40] Edward Forrest Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.

[41] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[42] Guillaume Salha, Romain Hennequin, Jean-Baptiste Remy, Manuel Moussallam, and Michalis Vazirgiannis. FastGAE: Scalable graph autoencoders with stochastic subgraph decoding. *Neural Networks*, 142:1–19, 2021.

[43] Hao Zhang, Bo Chen, Dandan Guo, and Mingyuan Zhou. WHAI: Weibull hybrid autoencoding inference for deep topic modeling. In *International Conference on Learning Representations (ICLR)*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Abstract
   (b) Did you describe the limitations of your work? [Yes] See Appendix
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1 and Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See supplemental material
   (b) Did you mention the license of the assets? [Yes] See supplemental material
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See supplemental material

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A The Pseudocode of Training Algorithm

Algorithm 1 The overall training algorithm of VEPM

Data: node features $\mathbf{X}$, observed edges $\mathbf{A}$, and observed labels $\mathbf{y}_o$.
Modules: Community ENCoder (CENC), Edge PARTitioner (EPART), Community-GNN BANK ($\mathrm{CGNN_BANK}$) and REPresentation COMPoser ($\mathrm{REPCOMP}$).
Parameters: $\boldsymbol{\omega}$ in the *inference network*, and $\boldsymbol{\theta}$ in the *generative network*.
Initialize $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$;
repeat
  $\boldsymbol{\Phi}, \mathbf{K}, \boldsymbol{\Lambda} \leftarrow \mathrm{CENC}(\mathbf{A}, \mathbf{X})$;
  compute $\mathcal{L}_{\mathrm{egen}}$ and $\mathcal{L}_{\mathrm{KL}}$, given in Equation (5);
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta_{\mathrm{unsup}} \cdot \nabla_{\boldsymbol{\omega}}(\mathcal{L}_{\mathrm{egen}} + \mathcal{L}_{\mathrm{KL}})$;
until CENC converges
repeat
  $\boldsymbol{\Phi}, \mathbf{K}, \boldsymbol{\Lambda} \leftarrow \mathrm{CENC}(\mathbf{A}, \mathbf{X})$;
  $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(K)} \leftarrow \mathrm{EPART}(\boldsymbol{\Phi}, \mathbf{A})$;
  for $step \leftarrow 1$ **to** M **do**
    $\hat{p}_{\mathbf{y}_o} \leftarrow \mathrm{softmax}\big(\mathrm{REPCOMP} \circ \mathrm{CGNN_BANK}(\boldsymbol{\Phi}, \mathbf{X}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(K)})\big)$;
    compute $\mathcal{L}_{\mathrm{task}}$, given in Equation (5);
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\mathrm{sup},\boldsymbol{\theta}} \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{task}}$;
  end for
  compute $\mathcal{L}$;
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta_{\mathrm{sup},\boldsymbol{\omega}} \cdot \nabla_{\boldsymbol{\omega}} \mathcal{L}$;
until the whole model converges

B Selection Rationale for the Probabilistic Distributions

The Bernoulli-Poisson distribution, which augments binary edges into counts of edges, not only adds interpretability in terms of edge partition, but also is well suited for sparse graphs. The gamma prior not only ensures positivity and promotes sparsity on Φ, but also allows an analytic KL term by combining with the Weibull variational distribution, which is reparameterizable. Here are some properties of the Weibull distribution [43] that we would like to highlight:

• **Similar density characteristics with gamma distribution**

The Weibull distribution has similar characteristics with the gamma distribution, *i.e.*, the density functions of the two distributions are similar, and both enjoy the flexibility in modeling sparse and nonnegative latent representations.

$$\begin{aligned} &\text{Weibull}(k, \lambda) \text{ PDF: } p(x|k, \lambda) = \frac{k}{\lambda^k} x^{k-1} e^{(x/\lambda)^k}, \\ &\text{Gamma}(\alpha, \beta) \text{ PDF: } p(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}. \end{aligned} \tag{6}$$

• **Easily Reparameterization**

The latent variable $x \sim \text{Weibull}(k, \lambda)$ can be easily reparameterized as

$$x = \lambda(-\ln(1-\varepsilon))^{1/k}, \;\; \varepsilon \sim \text{Uniform}(0, 1), \tag{7}$$

leading to an expedient and numerically stable gradient calculation.

• **Analytic KL-divergence**

Moverover, the KL-divergence between the Weibull and gamma distributions has an analytic expression formulated as

$$\begin{aligned}\mathrm{KL}(\mathrm{Weibull}(k,\lambda)||\mathrm{Gamma}(\alpha,\beta)) = -\alpha\ln\lambda + \frac{\gamma\alpha}{k} \\ +\ln k + \beta\lambda\Gamma(1+\frac{1}{k}) - \gamma - 1 - \alpha\ln\beta + \ln\Gamma(\alpha).\end{aligned} \tag{8}$$

C Experimental Settings

We evaluate VEPM on 11 benchmarks. For node classification, we consider three citation networks: Cora, Citeseer, and Pubmed, which provide bag-of-words document representations as node features and (undirected) citations as edges. For graph classification, we consider four bioinformatics datasets (MUTAG, PTC, NCI1, PROTEINS) and four social network datasets (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY and REDDIT-MULTI). The input node features are crafted in the same way as Xu et al. [6]. We summarize the statistics of these datasets in Table 6. For semi-supervised node classification, we use the train/test/validation split standardized by the GCNs [5] to make a fair comparison. We employ two protocols to evaluate the performance of VEPM on graph classification. Most of the baselines are compared following the 10-fold cross validation based evaluation protocol proposed by Xu et al. [6]. Since this protocol is criticized for being prone to overestimating model performance [30], we also evaluate our model following Zhang and Chen [31], who conduct a more rigorous train-validation-test protocol.

In the node classification experiments, for the *community encoder*, we adopt a similar encoder structure devised by Kipf and Welling [7], with the nonlinearity set to `softplus`. We select K, the number of metacommunities, through cross-validation, with $K = 4$ for Citeseer and $K = 8$ for Cora and Pubmed. We set the depth of the *generative network* to two layers, and evenly distribute the layers to the *community-GNN bank* and *representation composer*. Specifically, the *community-GNN bank* is implemented with K single-layer GCNs [5] and following Veličković et al. [13], we set their output dimensions as $64/K$. The *representation composer* is implemented with a single-layer GCN [5]. We record all hyperparameters in Table 7 in Appendix C.3.

For graph classification, we develop VEPM based on both GCN [5] and GIN [6], and preset the model hyperparameters when evaluating under the first protocol. We provide structural detials, the rationale behind hyperparameters determination in Appendix C.2 and sensitivity analysis in Appendix D. Through our studies, we find the variation of VEPM's graph classification performance with respect to different choices of model architecture or K is quite minor. For the second protocol, we follow the method of Zhang and Chen [31] to select hyperparameters, and report the test accuracy when the model achieves its highest validation accuracy over 100 epochs of training.

All experiments are carried out on NVIDIA Quadro RTX 5000, with Pytorch 1.9.0 and Python 3.8.

C.1 Data Statics

Table 6: Statistics of the datasets used for node- and graph-level classification tasks.

Task	**Node Classification**			**Graph Classification**							
Dataset	**Cora**	**Citeseer**	**Pubmed**	**IMDB-B**	**IMDB-M**	**MUTAG**	**PTC**	**NCI1**	**PROTEINS**	**RDT-B**	**RDT-M**
Graphs	1	1	1	1000	1500	188	344	4110	1113	2000	5000
Edges	5,429	4,732	44,338	96.5	65.9	19.8	26.0	32.3	72.8	497.7	594.8
Features	1,433	3,703	500	65	59	7	19	37	3	566	734
Nodes	2,708	3,327	19,717	19.8	13.0	17.9	25.5	29.8	39.1	429.6	508.5
Classes	7	6	3	2	3	2	2	2	2	2	5

C.2 Hyperparameter determination for the graph classification tasks

The model we develop for the graph classification tasks adopt a 2-layer GCN [5] for the *community encoder*. The dimension of ϕ is set to 100, and the dimension of the first hidden layer is set to 200 according to the 2:1 ratio in the number of neurons in the first two hidden layers of the GCN based encoder structure which is proved useful in node classification. We implement the *generative*

network based on GIN [6], which has four consecutive GIN layers whose hidden dimensions are all 64. Similar to our practice in VEPM for node classification, we evenly distribute the 4 layers to the *community-GNN bank* and the *representation composer*; in the *community-GNN bank*, the 64-dimension hidden units are evenly partitioned to all community-specific GNN components. Between the two choices of K we learned from the node classification experiments, we pick the smaller one, *i.e.*, $K = 4$ for the graph classification task, as the sizes of the graphs are much smaller than those for node classification. We conduct sensitivity analyses on the choices of K and the architectures of the *generative network*, and present the results in Appendix D.

C.3 All hyperparameters

Table 7: Hyperparameters settings for VEPM.

Hyperparameters	Experiments	
	Node Classification	Graph Classification
Community encoder	Settings	
α	1	1
β	1	1
epoches of *unsupervised pretrain*	1500	1500
learning rate of *unsupervised pretrain*	1e-3	1e-2
learning rate of *supervised finetune*	1e-3	1e-3
batch size	1	32
type of GNN layers	GCN	GCN
module structure	{32}-{16}	{200}-{100}
Edge partitioner	Settings	
number of metacommunities, K	[4,8]	4
τ for partition	0.1	1
community-GNN bank	Settings	
epoches of *supervised finetune*	200	100
learning rate of *supervised finetune*	1e-2	1e-3
weight decay of *supervised finetune*	5e-4	0.0
batch size of *supervised finetune*	1	32
concatenation weight of $\mathbf{\Phi}$	3e-2	3e-4
type of GNN layers	GCN	GIN
module structure	{64}	{64}-{64}
Representation composer	Settings	
epoches of *supervised finetune*	200	100
learning rate of *supervised finetune*	1e-2	1e-3
weight decay of *supervised finetune*	5e-4	0.0
batch size of *supervised finetune*	1	32
type of GNN layers	GCN	GIN
module structure	{#class}	{64}-{64}-{#class}

D Hyperparameter Sensitivity Analysis for Graph Classification

D.1 Number of metacommunities, K

To investigate the sensitivity of VEPM at different choices of K, the number of metacommunities, we hold all other hyperparameters as constants and run 10-fold cross validations with $K \in \{4, 5, 6, 7, 8\}$. From the results in Table 8, we can find that (i) the variation in graph classification performance that VEPM achieves with different values of K is not significant, most of the results recorded in the same column are within one standard deviation with each other; (ii) there is not a collective pattern between model performance and K shared across all benchmarks, in other words, not a single setting of K could consistently outperform other settings on all datasets. Given the above observations, we conclude that the performance of VEPM performance on graph classification is not significantly influenced by the number of metacommunities within a reasonable range.

Table 8: Comparisons of VEPMs with different numbers of metacommunities. * denotes the value we use in 10-fold cross validation.

# metacommunities	**IMDB-B**	**IMDB-M**	**MUTAG**	**PTC**	**NCI1**	**PROTEINS**
4*	76.7 ± 3.1	54.1 ± 2.1	93.6 ± 3.5	$\mathbf{75.6} \pm 5.9$	$\mathbf{83.9} \pm 1.8$	$\mathbf{80.5} \pm 2.8$
5	77.0 ± 1.8	52.9 ± 2.2	94.1 ± 6.3	72.4 ± 6.5	80.8 ± 2.2	78.1 ± 2.3
6	77.3 ± 3.2	53.9 ± 2.7	91.5 ± 6.4	73.5 ± 5.1	81.0 ± 3.0	75.8 ± 2.6
7	77.0 ± 2.5	$\mathbf{54.2} \pm 1.7$	$\mathbf{94.7} \pm 4.2$	71.8 ± 5.1	82.0 ± 1.3	79.0 ± 2.9
8	$\mathbf{78.3} \pm 2.1$	54.2 ± 2.1	92.5 ± 4.3	71.2 ± 3.7	78.3 ± 2.9	78.2 ± 2.9

D.2 Network Structure of the *generative network*

In this part, we focus on studying the potential influence of different network structures on the graph classification tasks. As we decide to make the *generative network* in VEPM to have a comparable number of parameters with GIN [6], the variation space left for the network structure is at how to distribute the 4 GIN layers to the *community-GNN bank* and *representation composer*. We use the name convention VEPM-$\{L_1\}$-$\{L_2\}$ to denote the variant with L_1 GIN layers in the *community-GNN bank* and L_2 GIN layers in the *representation composer*, where $L_1, L_2 \in \mathbb{Z}_+$ and $L_1 + L_2 = 4$. We hold all other hyperparameters as constants and evaluate VEPM under 3 structures: VEPM-$\{1\}$-$\{3\}$, VEPM-$\{2\}$-$\{2\}$ and VEPM-$\{3\}$-$\{1\}$. From the results shown in Table 9, the graph classification performance that VEPM achieves at different combinations of L_1 and L_2 are quite close to each other, the differences among the results in each column are at low statistical significance level. We hence conclude that the predictive power of VEPM is relatively stable at the variation of the network structure of the *generative network*.

Table 9: Comparisons of VEPMs with various network structures. * denotes the structure we adopt in 10-fold cross validation.

Network Structures	**IMDB-B**	**IMDB-M**	**MUTAG**	**PTC**	**NCI1**	**PROTEINS**
VEPM-$\{1\}$-$\{3\}$	$\mathbf{77.8} \pm 2.0$	$\mathbf{56.3} \pm 2.1$	91.0 ± 4.9	73.3 ± 3.6	81.2 ± 2.5	77.4 ± 3.6
VEPM-$\{2\}$-$\{2\}^*$	76.7 ± 3.1	54.1 ± 2.1	$\mathbf{93.6} \pm 3.5$	$\mathbf{75.6} \pm 5.9$	83.9 ± 1.8	$\mathbf{80.5} \pm 2.8$
VEPM-$\{3\}$-$\{1\}$	74.5 ± 2.2	54.3 ± 3.2	87.0 ± 6.8	66.3 ± 5.0	$\mathbf{84.0} \pm 2.7$	78.9 ± 3.2

E Subsampling-Based Acceleration Algorithm

Like all neural graph generative models that model the connection statuses of all $\binom{N}{2}$ node pairs, *e.g.*, VGAE [7] and its various extensions, the time complexity of (pre)training VEPM is also $\mathcal{O}(N^2)$. Thus a direct implementation that considers all node pairs in each iteration will face the scalability issue as N increases. Fortunately, such computation burden could be reduced to $\mathcal{O}(N_{\mathcal{S}}^2)$ through subgraph-sampling based acceleration, where $N_{\mathcal{S}}$ denotes the number of nodes in the subgraph after sampling, as analogous to FastGAE [42]. Specifically, we develop an explainable and flexible strategy

to sample nodes from the original graph *with replacement*, where node i is sampled at probability p_i:

$$p_i = kq_i + (1-k)\frac{1-q_i}{N-1},\ q_i = \frac{f(\mathbf{x}_i)^\alpha}{\sum_{j=1}^{N}(f(\mathbf{x}_j)^\alpha)}. \tag{9}$$

With $f(\cdot)$ measuring the "importance" of node i, q_i denotes the probability of sampling node i for its relative importance in graph $\mathcal{G}$. The sharpness of the probability distribution $\{q_i\}_{i=1,N}$ is adjusted by the exponent α, where greater sharpness can be obtained with a higher value of α, and $\{q_i\}_{i=1,N}$ would be equivalent to an uniform distribution if α is set to 0. Likewise, $\frac{1-q_i}{N-1}$ can be interpreted as the probability that node i is sampled for its relative unimportance. For a better coverage, both "important nodes" and "unimportant nodes" are to be included in the sampled subgraph, with their percentages balanced by $k \in [0, 1]$, *i.e.*, the higher the value of k, the more "important nodes" are expected to be included in the subgraph. We use $k = 0.9$, $\alpha = 1$, and node degrees as $f(\cdot)$.

We record both node classification accuracy and the elapsed time per epoch for VEPM trained with and without subgraph sampling. The modeled is evaluated on Cora, Citeseer and Pubmed where $N_{\mathcal{S}}$ is set to 200. As shown in Table 10, the acceleration algorithm successfully reduces the time cost by a huge margin, it also results in a slight decrease in model performance and a mild inflation in the error bar. However, the negative impacts are not significant enough to offset the achievement in scalability, especially considering the fact that with the implication in model performance, VEPM with scalable training still outperforms other baselines on two out of three benchmarks.

Table 10: Comparison of VEPMs with or without scalable training.

Hyperparameters	Cora		Citeseer		Pubmed	
	Accuracy (%)	Epoch duration	Accuracy (%)	Epoch duration	Accuracy (%)	Epoch duration
Full-batch Training	**84.3** $\pm$ 0.1	0.19s	72.5 $\pm$ 0.1	0.25s	**82.4** $\pm$ 0.2	7.28s
Scalable Training	83.4 $\pm$ 0.1	0.02s	71.8 $\pm$ 0.3	0.03s	81.6 $\pm$ 0.3	0.05s

F More Ablation Studies

Different edge partition. In this part, we study how much a meaningful edge partition would benefit the task. Except for the partition we obtained from training the regular VEPM, we tested the performance of VEPM under two edge partition schemes: even partition and random partition. Even partition means all edges weights are $1/K$. For random partition, we sample the unnormalized edge weights from $\mathrm{U}(0, 100)$, then normalize them along the community dimension with `softmax`. The random edge weights are fixed for training to simulate an undesirable convergence state of edge partition. The results are recorded in Table 11. Generally speaking, the performance of VEPM with even edge partition is comparable to that of GIN, our base model; when we substitute random partition for even partition, the model performance becomes worse than the base model. These ablation study results suggest that VEPM can benefit from enhancing the quality of edge partitions.

Table 11: The performance of VEPM with different edge partition schemes. The suffix "-EvP" stands for even edge partition, and "-RdP" stands for random edge partition.

Method	IMDB-B	IMDB-M	MUTAG	PTC	PROTEINS
VEPM-EvP	74.2 $\pm$ 4.3	51.4 $\pm$ 2.7	90.4 $\pm$ 5.4	66.3 $\pm$ 6.6	76.3 $\pm$ 3.2
VEPM-RdP	60.5 $\pm$ 4.7	41.1 $\pm$ 1.5	81.4 $\pm$ 5.6	61.8 $\pm$ 4.0	62.7 $\pm$ 3.8
GIN	75.1 $\pm$ 5.1	52.3 $\pm$ 2.8	89.4 $\pm$ 5.6	64.6 $\pm$ 7.0	76.2 $\pm$ 2.8
VEPM	76.7 $\pm$ 3.1	54.1 $\pm$ 2.1	93.6 $\pm$ 3.4	75.6 $\pm$ 5.9	80.5 $\pm$ 2.8

Different *representation composer*. Next, we use the partition obtained from a regular VEPM to represent partitions that are meaningful for the task, and a random partition to represent "meaningless partitions." For each partition scheme, we contrast the performance of VEPM between two designs of the *representation composer*: (i) GNN-based design, represented by the current design; (ii) parsimonious design, represented by a fully-connected (FC) layer. The results are in Table 12. When the edge partition is meaningful, *i.e.*, relevant to the task, the overall performance with a simpler *representation composer* is generally slightly worse than the current design, but still better than most

of the baselines. However, if the partition is irrelevant to the task, the GNN-based *representation composer* would be the last thing standing between the model and a failed task, as the graph structure used for neighborhood aggregation is still informative to the task. So the current design would slightly benefit VEPM in a general case, and protects the performance of the model in the worst-case scenario.

Table 12: The performance of VEPM with different designs of the *representation composer*.

	representation composer design	**IMDB-B**	**IMDB-M**	**MUTAG**	**PTC**	**PROTEINS**
meaningful partition	GNN-based	76.7 ± 3.1	54.1 ± 2.1	93.6 ± 3.4	75.6 ± 5.9	80.5 ± 2.8
	FC-based.	74.6 ± 3.4	51.6 ± 1.7	90.3 ± 4.2	68.7 ± 4.3	76.7 ± 3.2
meaningless partition	GNN-based	60.5 ± 4.7	41.1 ± 1.5	81.4 ± 5.6	61.8 ± 4.0	62.7 ± 3.8
	FC-based.	58.6 ± 3.8	45.3 ± 2.0	78.6 ± 5.9	58.5 ± 5.9	59.8 ± 3.9

G The Working Mechanism of VEPM

In this section we explore how the communities benefit the graph-analytic task in VEPM, and we draw the conclusions from two exploratory experiments on the Cora dataset. We first assess the relevance of the learned communities to the task. In order to see the statistical dependency between communities and the task, we first hard-assign each node to the community of the highest affiliation strength, then we compute the normalized mutual information (NMI) between the hard-partitioned communities and node labels. NMI is a score between 0 and 1, with higher values indicating stronger statistical dependencies between two random variables. In the above experiment, we obtain an NMI of 0.316 when the communities are learned solely by the *unsupervised pretrain*, such value increases to 0.322 after the *supervised finetune*. The results of this experiment show that the initial communities we obtained from the *unsupervised pretrain* are meaningful to the task, and the relevance between the inferred communities and the task would be enhanced by the *supervised finetune*.

Next, we explore how the information provided by each community facilitates the task. To see the effect of each community, we first train VEPM until convergence, then use the node representations learned by each community GNN to train an SVM and visualize the averaged 10-fold cross-validation results by a confusion matrix. With the results shown in Figure 5, we can further conclude that most of the communities can provide task-related information, as we can observe from Figure 5 that the diagonal cells in most of the confusion matrices have darker shades than off-diagonal cells. Beyond that, the information extracted from different communities is complementary to each other. For example, in the second column of Figure 5, information from the upper community can separate classes 3 and 4 well, but would mix classes 4 and 5; information from the lower community, on the other hand, can separate classes 4 and 5 but works worse on separating classes 3 and 4.

In summary, the communities learned by VEPM have the following properties: (i) the communities are relevant to the task; (ii) the information provided be different communities are complementary, so the overall amount of information for the task is accumulated over communities. Both properties are intuitively helpful for the downstream task.

H Limitations and broader impacts

While our model exhibits superiority over traditional convolution based graph neural networks on both node-level and graph-level representation learning tasks, several caveats are noteworthy. From the perspective of model design, we only focus on intra-community node interaction in our graph generative model, while it is a common practice and usually leads to good results in modeling homophilous graphs, it cannot well explain the edges generated as a result of cross-community node interaction, which potentially limits its modeling performance on heterogeneous graphs.

VEPM offers a a new perspective to model the network with latent communities, it separates latent communities from the entire network and models them individually. Used with goodwill, it can understand what optimally benefits each community and help policy-making accordingly. However, along with its modeling power that could be harnessed to benefit communities, it can be dangerous if been put into socially harmful applications. After all, no model is self-immune to malicious purpose,

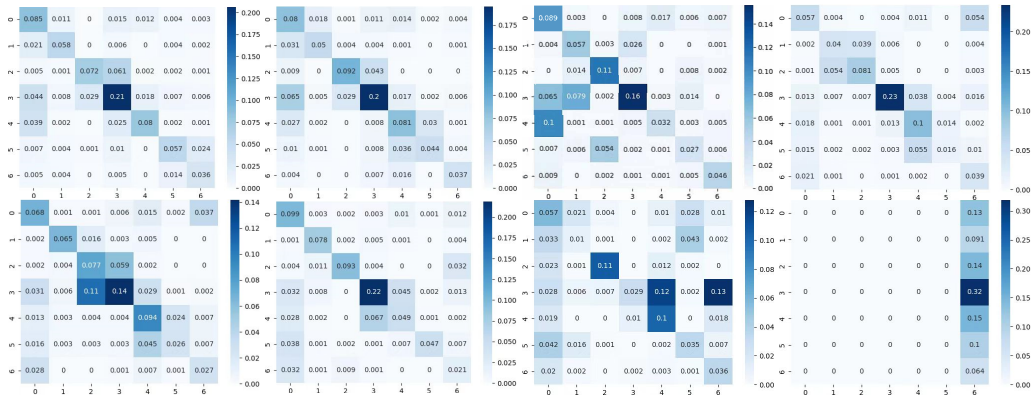


Figure 5: The 10-fold cross-validation results of using node representations learned from each of the eight communities for node classification, expressed in the format of normalized confusion matrices. The cell at the intersection of the ith row and jth column is the percentage of (mis)classifications that predict the label of an ith class node as j. A darker shade indicates a higher (mis)classification rate. Please note that the same color for different confusion matrices may correspond to different values.

the potential misusages of VEPM include but not limited to group-targeted broadcast of fake news or conspiracy theories.

I Partitioned Adjacency Matrix

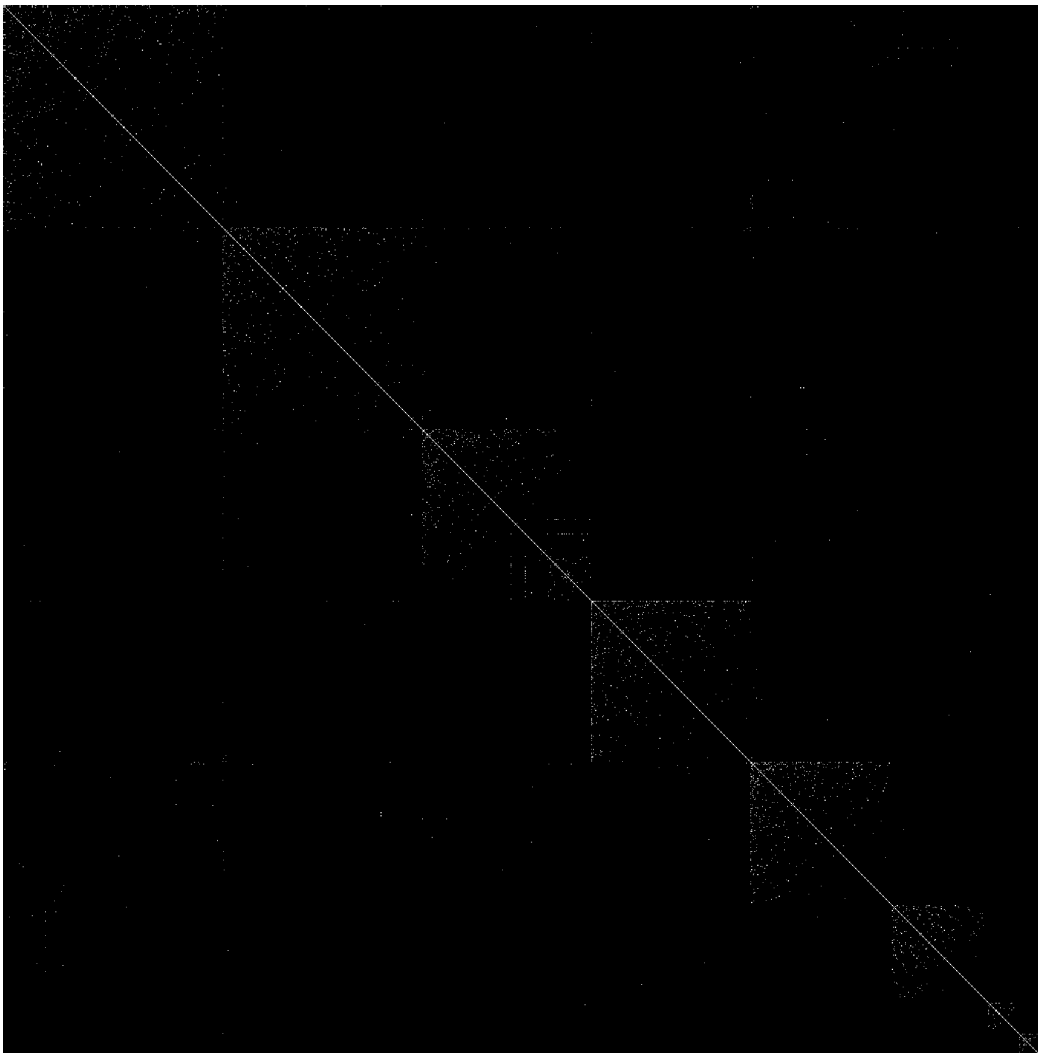


Figure 6: $\mathbf{A}$, the entire adjacency matrix of the Cora dataset.

Figure 7: $\mathbf{A}^{(1)}$, the partitioned graph corresponding to the largest metacommunity.

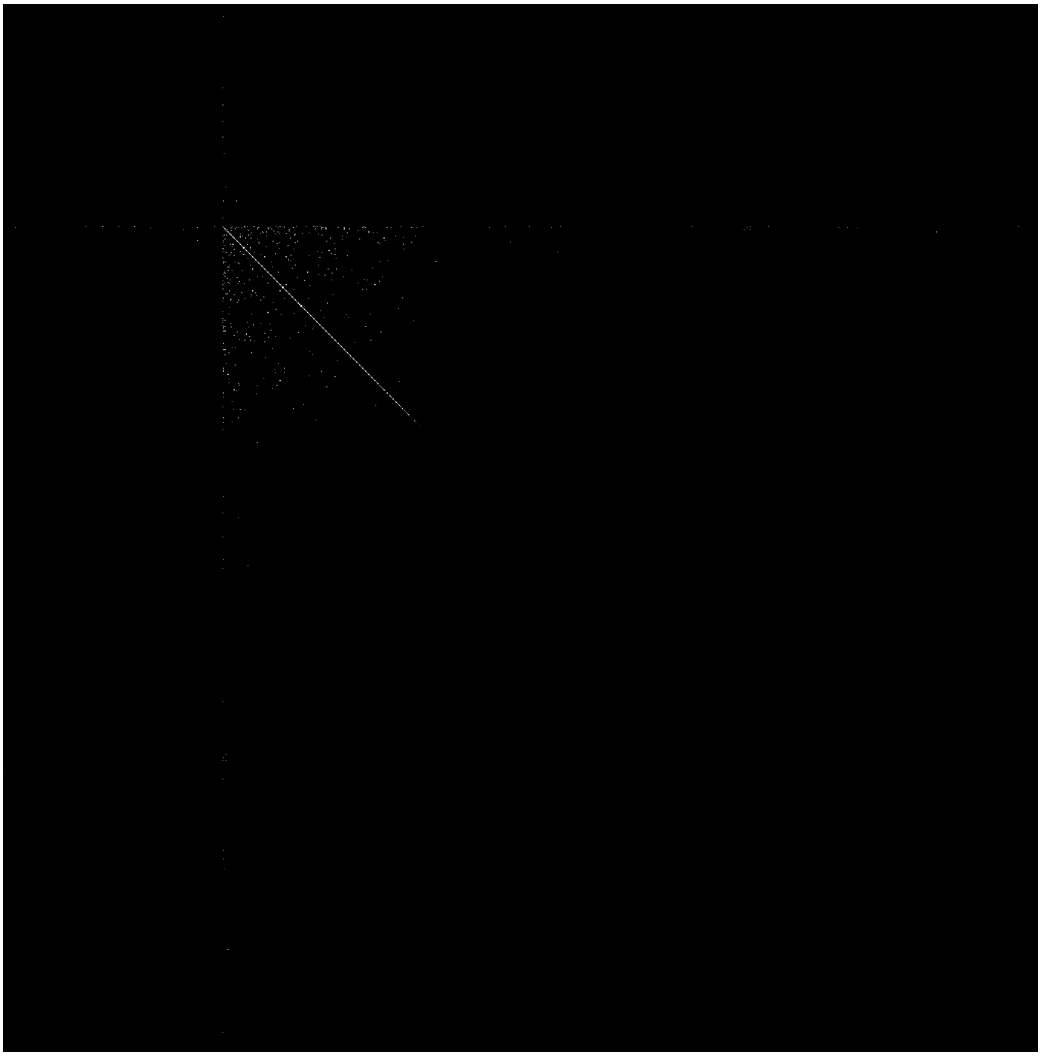


Figure 8: $\mathbf{A}^{(2)}$, the partitioned graph corresponding to the second largest metacommunity.

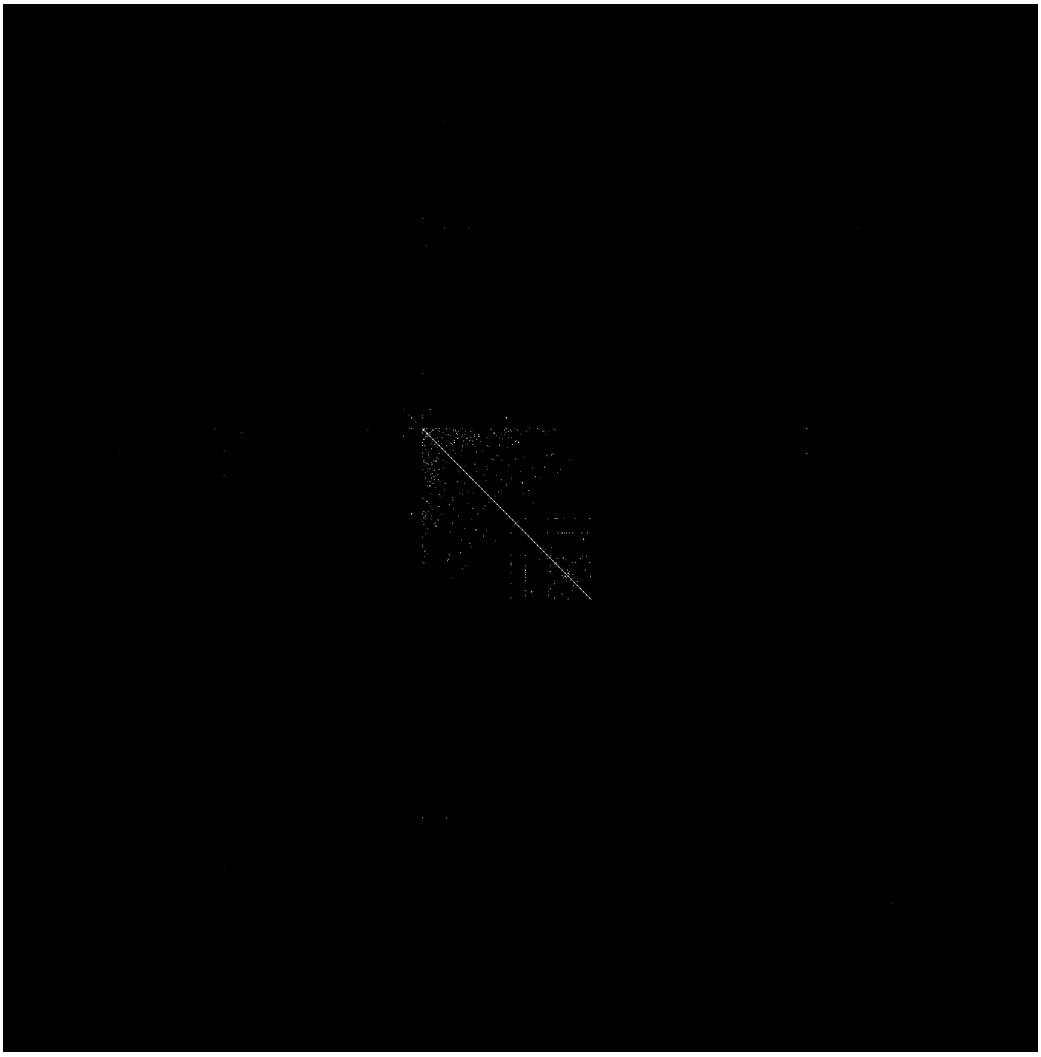


Figure 9: $\mathbf{A}^{(3)}$, the partitioned graph corresponding to the third largest metacommunity.

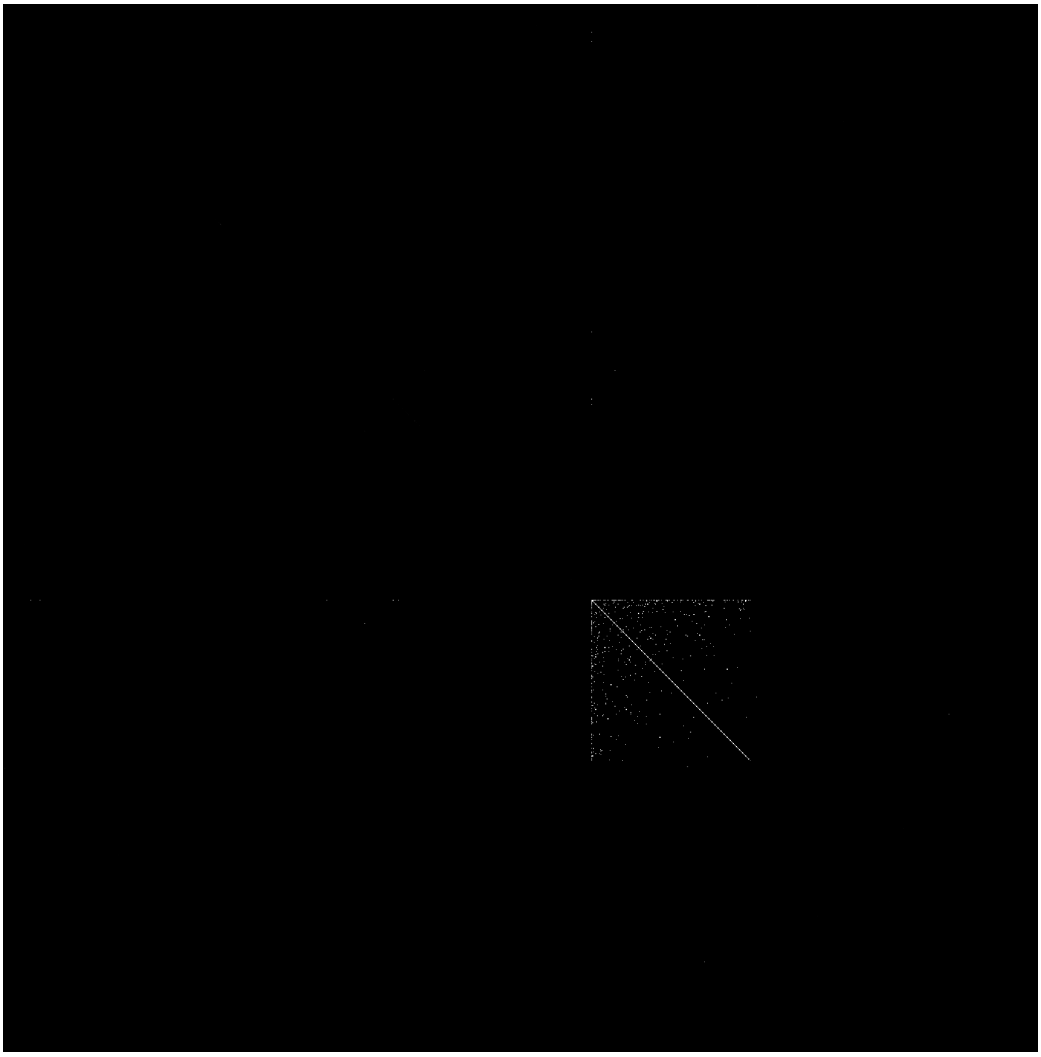


Figure 10: $\mathbf{A}^{(4)}$, the partitioned graph corresponding to the fourth largest metacommunity.

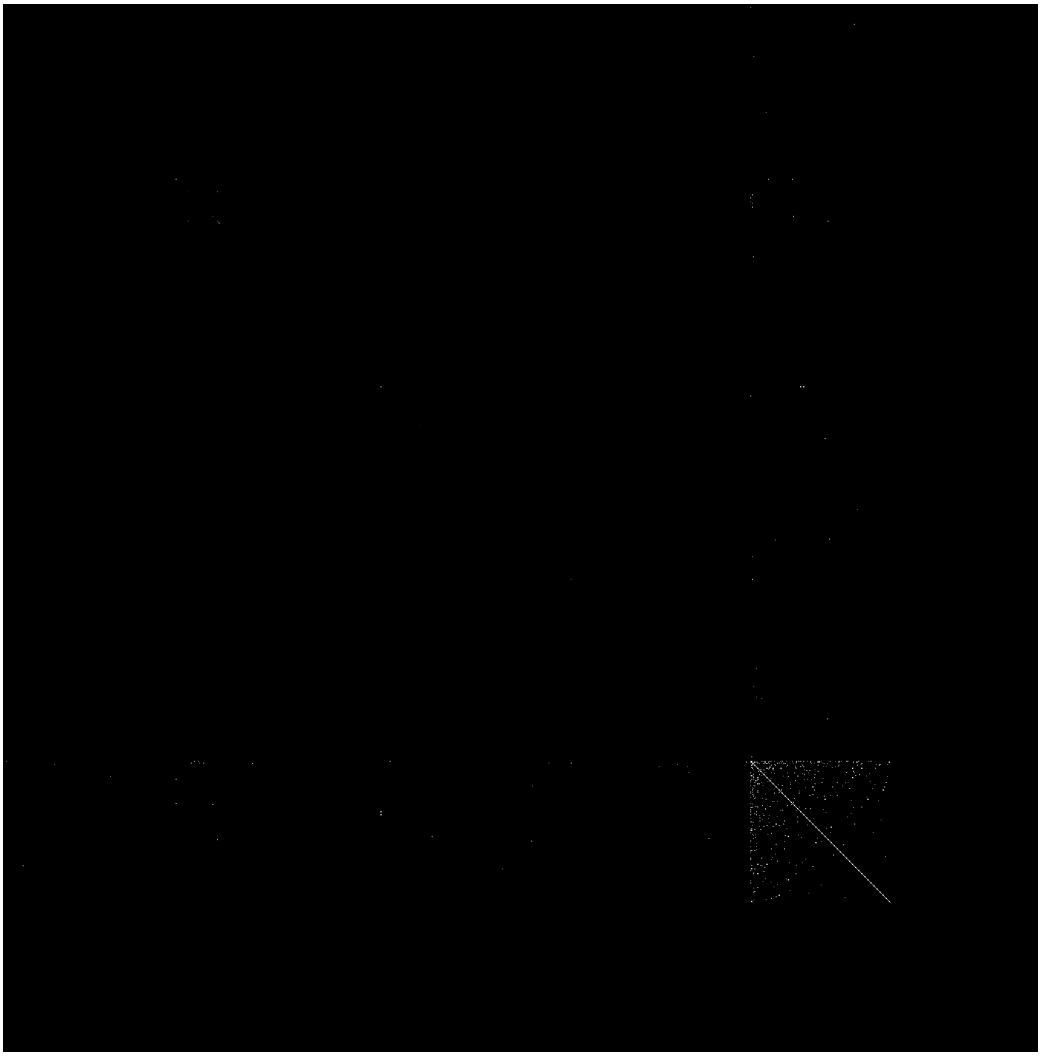


Figure 11: $\mathbf{A}^{(5)}$, the partitioned graph corresponding to the fourth smallest metacommunity.

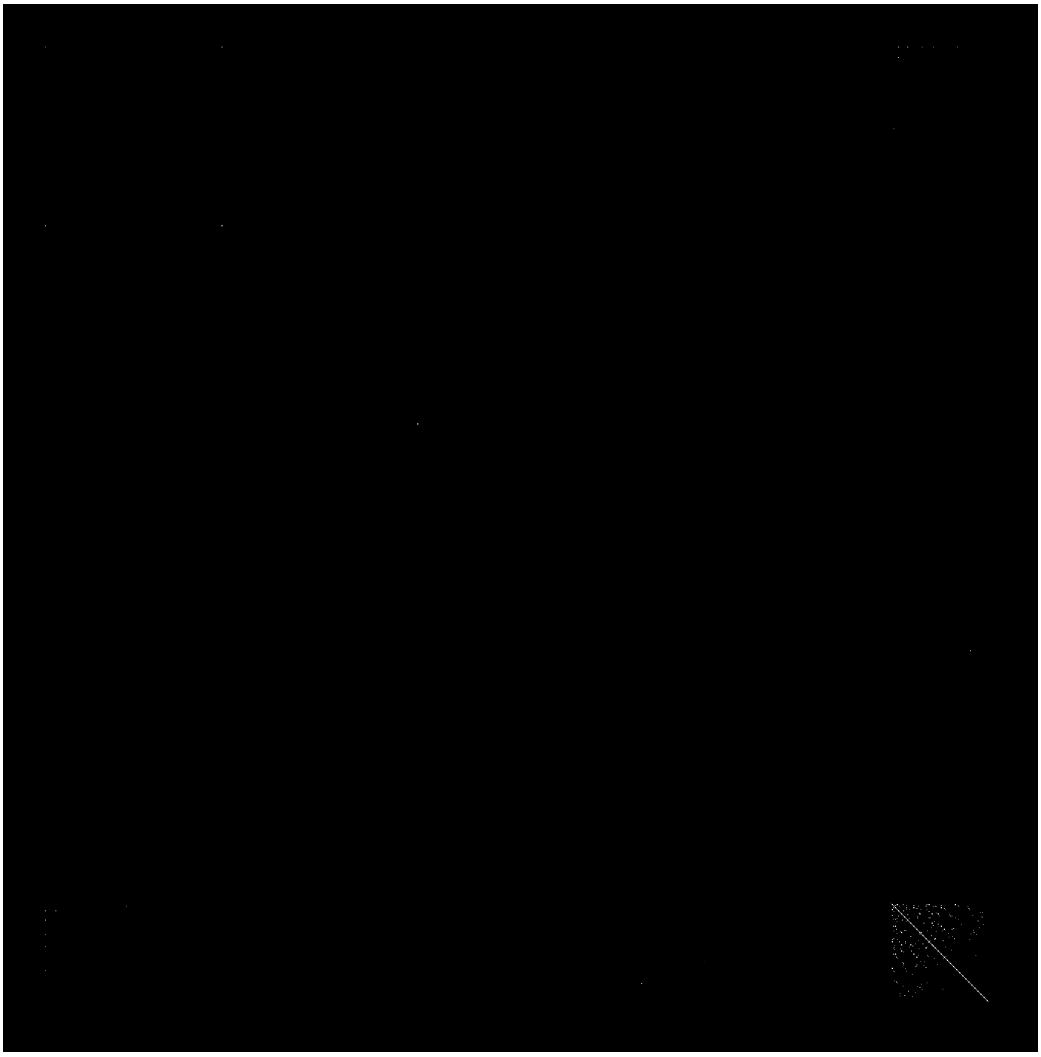


Figure 12: $\mathbf{A}^{(6)}$, the partitioned graph corresponding to the third smallest metacommunity.

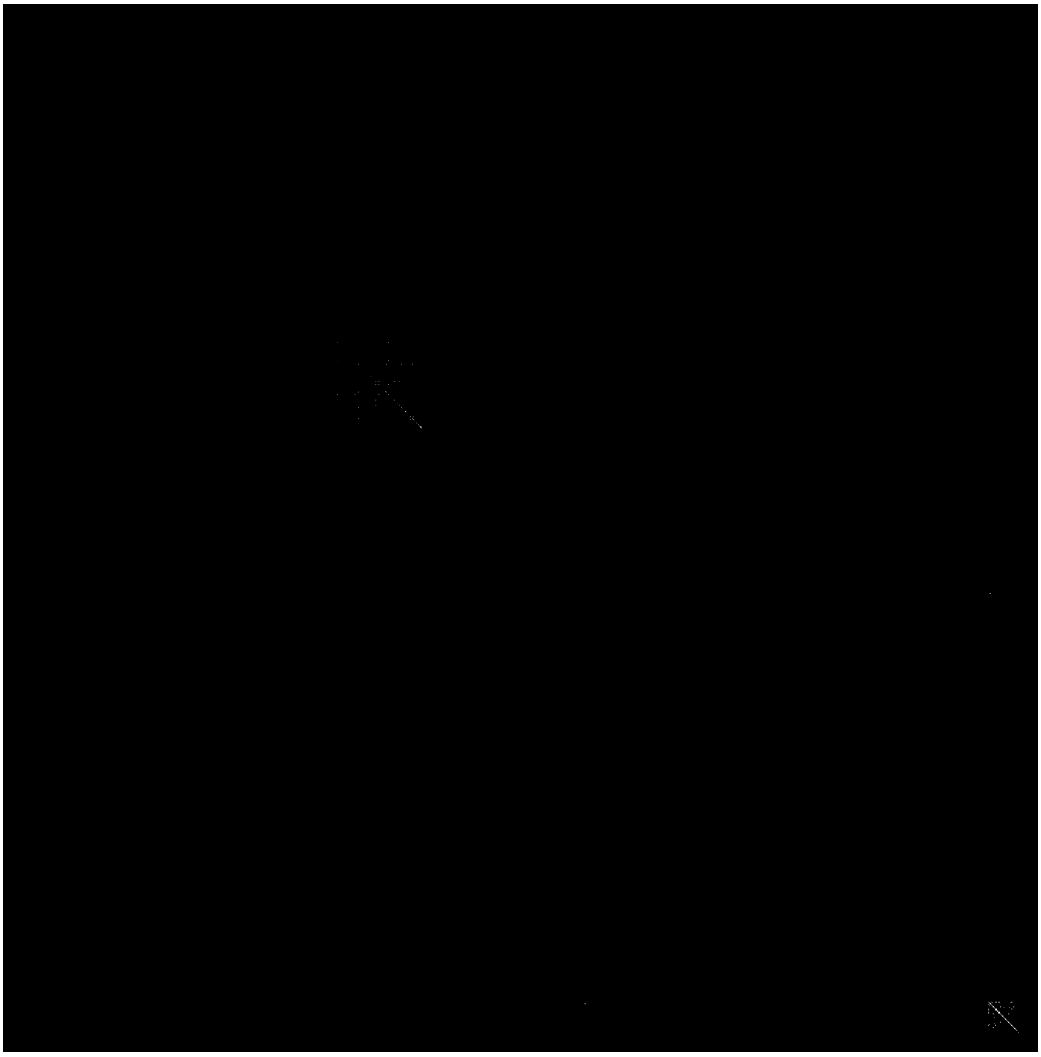


Figure 13: $\mathbf{A}^{(7)}$, the partitioned graph corresponding to the second smallest metacommunity.

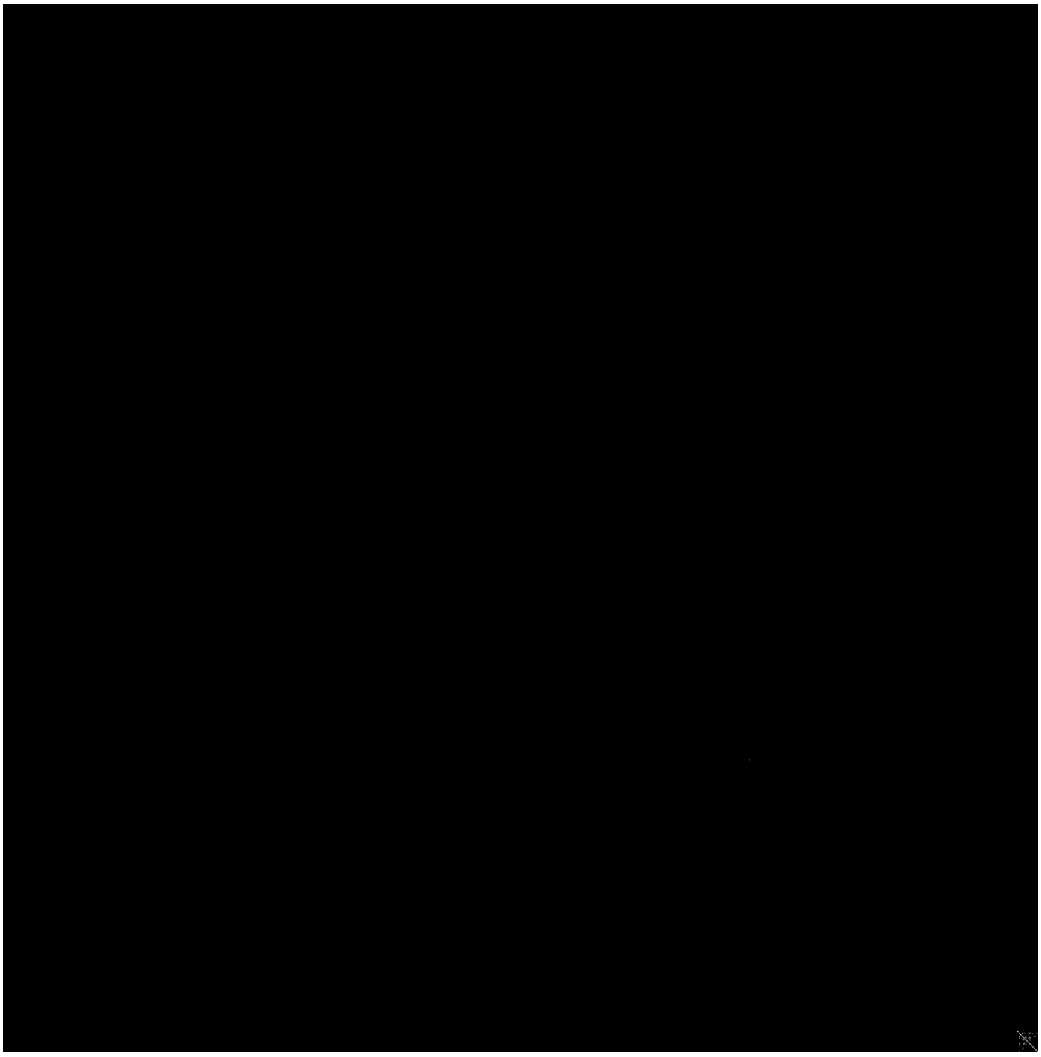


Figure 14: $\mathbf{A}^{(8)}$, the partitioned graph corresponding to the smallest metacommunity.