Manuscript Title: with Forced Linebreak*

Ann Author† and Second Author‡
Authors' institution and/or address
*This line break forced with *
(MUSO Collaboration)

Charlie Author§
Second institution and/or address
This line break forced and
Third institution, the second for Charlie Author

Delta Author
Authors' institution and/or address
*This line break forced with *
(CLEO Collaboration)
(Dated: June 30, 2022)

Because of its powerful ability to extract fundamental features from complex systems, deep learning has a wide range of applications in areas such as image recognition, speech processing, and biological systems. In particular, in the field of condensed matter physics, deep learning has demonstrated the unique ability to accurately approximate complex physical systems. In the past few years, traditional neural networks have achieved good results in condensed matter physics both in terms of feature extraction and model generation. However, the limitation of configuration size has led to the generation of only single-size configurations, which limits the generalizability of neural networks. Traditional deep learning algorithms assume that data samples are independent of each other, but in real physical configurations, each spin node is associated with other nodes and edges in the configuration. This problem is solved by the emergence of graph neural networks, which are irregular, can adapt to arbitrary size configurations, and can capture the interdependencies between different nodes. Variational self-coding is an unsupervised learning algorithm that can effectively model the distribution of training data. In this paper, we use this feature of graph neural networks in combination with variational self-coding to construct Ising model simulators, and the generated Ising models are very similar to the data simulated by traditional Monte Carlo methods. The results show that graph neural networks are a promising computational tool in the field of condensed matter physics.

I. INTRODUCTION

Over the past year, physicists have started to use deep learning techniques to study the field of condensed matter physics. Most of the tasks are done by supervised learning algorithms. In supervised learning, the algorithm is trained on data with labels, which are assigned to data points. After successful training, high precision predictions can be made for labels of previously unseen data. In addition to supervised learning, there are unsupervised learning algorithms that can find structure in unlabeled data. It has become possible to use unsupervised learning techniques to reproduce the Monte Carlo sampling state of the Ising model, and discovering phase transitions in an unsupervised manner has become mainstream. Restricted Boltzmann machines, variational self-coding, and generative adversarial networks have been widely used for the simulation of Ising models with good results. However, none of these models can take into account the molecule-molecule interactions and the influence of the edges between molecules in the physical model during training. Also, the microstructure of many physical models is irregular, and the generated models are poorly generalized and can generally only accommodate one size at one temperature. This requires a deep neural network that can consider unstructured data and can take into account node-to-node and node-to-edge relationships.

Fortunately, the research on graph neural networks in the field of deep learning has been growing in enthusiasm in recent years, and the excellent ability of graph neural networks to handle unstructured data makes them particularly useful for applications in physics, biology, and chemistry. Most physical models are essentially a graph structure. Compared with traditional convolutional networks, graph convolutional networks are better at encoding structural information of graphs and can learn better and deeper representation features. Graph vari-

* A footnote to the article title
† Also at Physics Department, XYZ University.
‡ Second.Author@institution.edu
§ http://www.Second.institution.edu/~Charlie.Author

ational self-coding is an unsupervised graph generation algorithm, which is the product of combining graph neural network and generative network, and it can effectively use the graph structure of data to simulate the distribution of training data.

In this paper, we integrate a graph convolutional network into a graph variational self-encoder framework to construct an Ising model simulator, which is used to simulate Ising model configurations at different temperatures. The trained simulator can effectively generate Ising model states with physical properties that are not different from those obtained from conventional Monte Carlo simulations. It can also be adapted to a wide range of different sizes and significantly reduces the simulation time. We will first introduce the general form of the Ising model, graph variational neural networks and graph convolutional networks. In the core of the paper, we construct a graph variational self-encoder combined with a graph convolutional network to simulate the 2D Ising model at different temperatures and verify the validity of our method by comparing the generated data with those from Monte Carlo simulations.

II. ISING MODEL AND GRAPH NEURAL NETWORKS

A. Ising model

In statistical physics, the Ising model is described as the set of binary spins with coupling interactions in some lattice. Considering that N spins s=s_i can take the value $\pm$ 1, where the index i marks the position of the spin s_i, the standard Hamiltonian of the Ising system includes only the nearest-neighbor interactions, and each spin direction may be "up" (+1) or "down" (-1), although the generalized model may include long-range interactions and more spin-selective directions. The standard Hamiltonian quantities are:

$$H = -J \sum_{neighbors} S_i * S_j \tag{1}$$

B. Graph neural network

A graph is a data structure that models a set of objects (nodes) and their relationships (edges). This year, research on analyzing graphs with machine learning methods has received increasing attention due to the powerful expressiveness of graph structures. Graph neural network is a general term for algorithms that use neural networks to learn graph structured data, extract and uncover features and patterns in graph structured data, and meet the needs of graph learning tasks such as clustering, classification, prediction, segmentation, and generation. One motivation for the development of graph neural networks originated from convolutional neural networks. The widespread use of convolutional neural networks has led to breakthroughs in machine learning and opened the era of deep learning. However, convolutional neural networks can only extract potential features in regular Euclidean space data, which cannot correspond well to the complex and variable graph data in reality, and how to apply convolutional neural networks to the non-Euclidean space of graph structure has become the key problem of graph neural network models. The descriptions and definitions of the relevant symbols in graph neural networks are given below

Definition 1 *Graph:*The graph is formed by the nodes and the edges connecting the nodes, usually noted as $G = (V, E)$. where $V = \{v_1, v_2, ..., v_n\}$ represents the set of nodes,$E = \{e_1, e_2, ..., e_n\}$ represents the set of edges, and edges can also be represented as (v_1, v_2). Usually nodes are also called vertices or intersections. Edges are also called links or arcs. A generic graph representation is a quintet: $G(V, E, A, X, D)$. Where A represents the adjacency matrix of the graph, X represents the identity matrix of the nodes, and D represents the degree matrix.

Definition 2 *Adjacency matrix*: The adjacency matrix of a graph refers to the matrix used to represent the connectivity of the nodes in the graph. This matrix can be binary or weighted. For an undirected graph with N nodes, the adjacency matrix is an N*N real symmetric matrix.

Definition 3 *Degree matrix*: The degree of a node represents the number of edges connected to that node. The degree matrix of a graph is the matrix that describes the degree of each node in the graph. The degree matrix is a diagonal matrix, and for undirected graphs, only the incoming degree matrix or the outgoing degree matrix is generally used.

Definition 4 *Combinatorial Laplacian matrix*, also known as standard Laplacian matrix, is a combination of diagonal matrix and adjacency matrix:

$$L = D - A \tag{2}$$

The matrix has non-zero elements only at the central node and at the nodes connected to the first order, and zero everywhere else. Laplacian matrix is also a form of graph.

Definition 5 *Normalized Laplacian matrix*

$$L^{sym} = I - D^{-1/2} A D^{-1/2} \tag{3}$$

Its element value is :

$$L^{sym}_{(i,j)} = \begin{cases} 1, & \text{i=j,} deg(v_i) \neq 0 \\ \frac{-1}{\sqrt{deg(v_i) deg(v_j)}}, & i \neq j \text{ ,} v_i \text{ and } v_j \text{ are adjacent} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $deg(v)$ is denoted as the degree of node v

1. Graph Convolutional Network

A graph convolution network is derived from the traditional convolutional neural network. Graph convolution can be of two kinds, spectrum-based approach and space-based approach. The spectrum-based approach defines graph convolution from the perspective of graph signal processing by introducing filters, so the spectrum-based graph convolution can be understood as removing noise from the graph signal. The space-based graph convolution method constructs the graph convolution by pooling information from neighboring nodes.

Before understanding graph convolution, we need to have a basic concept of convolution operation. Convolutional networks are first applied to images, where we use the pixel points on the image as features to perform the convolution operation to extract features. The random shared convolution sum is used to obtain a weighted sum of pixel points to extract a particular feature, and then back propagation is used to optimize the convolution kernel parameters to automatically extract features, which is the cornerstone of convolutional neural network to extract features. The convolution operation is equivalent to aggregating the information of surrounding pixels to the central pixel of the convolution kernel and extracting the features of that part.

So we want to use convolution in the graph domain to use a general paradigm for graph feature extractionm, As shown in FIG. 1, when we want to get the feature representation of a node, we can aggregate the information of neighboring nodes, which can be understood in this way, each node in the graph is changing its state all the time until the final balance because of the influence of the more distant points of the neighboring node kernel, the closer the neighbor The closer the neighbor, the greater the influence.

2. Graph Convolutional Generalization

For a graph convolutional network, we want to learn its feature representation for each node, and any graph convolution can be written as a nonlinear function such that:

$$H^{l+1} = f(H^l, A) \tag{5}$$

$H^0 = X$ is the input to the first layer, l represents the number of layers of the neural network, and A represents the adjacency matrix.

The traditional graph convolutional network approach is to add information from neighboring nodes to this node:

$$f(H^l, A) = \sigma(AH^lW^l) \tag{6}$$

W^l represents the parameter matrix of the lth layer, and σ represents the activation function. According to the matrix multiplication it can be seen that each node in the above equation combines the information of the neighboring nodes, but since only the adjacency matrix is used, the diagonal of the adjacency matrix is 0, which cannot reflect the information of the nodes themselves, so we use the Laplacian matrix instead of the adjacency matrix

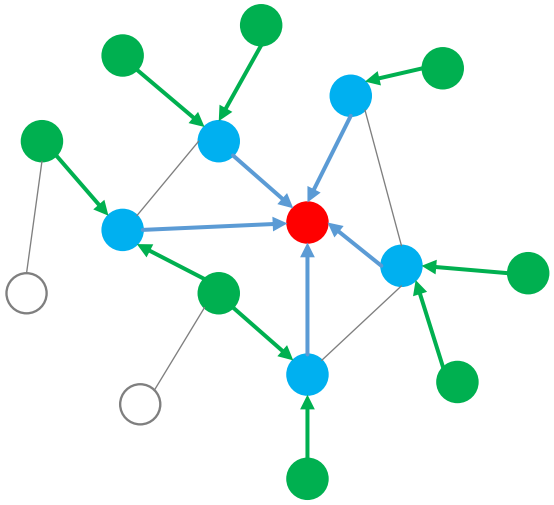


FIG. 1: Schematic diagram of a graph convolutional network aggregating neighboring nodes. The red node in the center aggregates the nearest blue node, and the blue node in turn aggregates the nearest green node, allowing the nodes to gain longer-range influence through multiple sets of convolutional operations.

$$f(H^l, A) = \sigma(LH^lW^l) \tag{7}$$

The Laplace matrix is introduced in the above equation(7), thus solving the problem of not considering the information self-transfer of its own nodes, but since it is not normalized, we regularize the Laplace matrix to obtain the graph convolution

$$f(H^l, A) = \sigma(L^{sym}H^lW^l) \tag{8}$$

The above description is in matrix form, we can also look at the formula from the perspective of a single node, for the node feature h_i^{l+1} in the $l+1$ layer, for his neighboring nodes $j \in N$, N is the set of neighboring nodes used by node i, so the update formula of the graph neural network can also be depicted as

$$h_i^{l+1} = f(h_i^l) = \sigma(h_i^l \bullet W_1^{l+1} + \sum_j h_j^l \bullet W_2^{l+1}) \tag{9}$$

3. Variational auto-encoder

A variational self-encoder is a generative model of variational inference, consisting of a network of encoders and decoders, which is a model with hidden variables. In a

variational self-encoder, we base on the following assumptions: Our sample x is generated by some latent variable z through some mapping, and z is not a fixed value, but obeys a distribution: $z \sim P_\theta(z)$, then $x \sim p_\theta(x|z)$, where P is a family of distributions determined by the parameter θ, and θ is the true distribution that we need to find. So our fundamental goal is to determine the values of z and θ so that we can obtain the sample x. In order to infer $p_\theta(x|z)$, we need to maximize the marginal log likelihood $log(p_\theta(x))$ We can rewrite the equation:

$$\begin{aligned} log(p_\theta(x)) &= log \int p_\theta(x|z)dz = log \int p_\theta(x|z)p(z)dz \\ &= log \int p_\theta(x|z)p(z)\frac{q_\phi(z|x)}{q\phi(z|x)}dz \\ &\geq \mathbb{E}_{z\sim q_\phi(z|x)}[log(p_\theta(x|z)] - D_{KL}[q_\phi(z|x)||p(z)] \\ &:= \varepsilon(x,\theta,\phi) \end{aligned} \tag{10}$$

We apply Jensen's inequality in the fourth step. Integrating the potential variable z in the second step is usually tricky, so we introduce an approximate posterior distribution $q_\phi(z|x)$ with the parameter set ϕ and use the variational inference principle to obtain an easy-to-handle bound on the marginal log-likelihood, i.e., the evidence lower bound (ELBO). We use $p(z) = N(1,0)$ as the latent variable prior. ELBO is an easy-to-process lower bound on the log-likelihood, and thus maximizes the inference $p_\theta(x|z)$. $\mathbb{E}_{z\sim q_\phi(z|x)[log(p_\phi(x|z))]}$ can be interpreted as the reconstruction error, since maximizing it will make the output of the decoder similar to the input of the encoder, and $D_{KL}[q_\phi(z|x)||p(z)]$ is the KL scatter (see *** for details), a value that measures the similarity of two distributions and that ensures that the latent representation is Gaussian, such that data points with similar characteristics have similar Gaussian representations.

We have outlined the general idea behind latent variable models, but we still need to specify the approximate (variational) posterior $q_\phi(z|x)$ and the model likelihood $log(p_\theta(x|z))$. A common choice for approximating the posterior is to decompose the Gaussian distribution

$$q_\phi(z|x) = \prod_{j=1}^{d} N(z_j|\mu_j, \sigma_j^2) \tag{11}$$

where μ and σ denote the mean and standard deviation of the model.

In summary, VAE is an encoder network based on extracting the mean and variance of a potential Gaussian representation $q_\phi(z|x)$ of $p(z|x)$. The decoder in VAE, which uses a Gaussian $q_\phi(z|x)$ sample as input, generates new samples according to the distribution $p_\theta(x|z)$(see FIG.3). We compute all parameters by maximizing ELBO using backpropagation, but the Gaussian distribution is random and non-integrable. Therefore a deterministic and differentiable mapping between the output of the encoder and the input of the decoder needs to be established As shown in the FIG.2. We need to represent

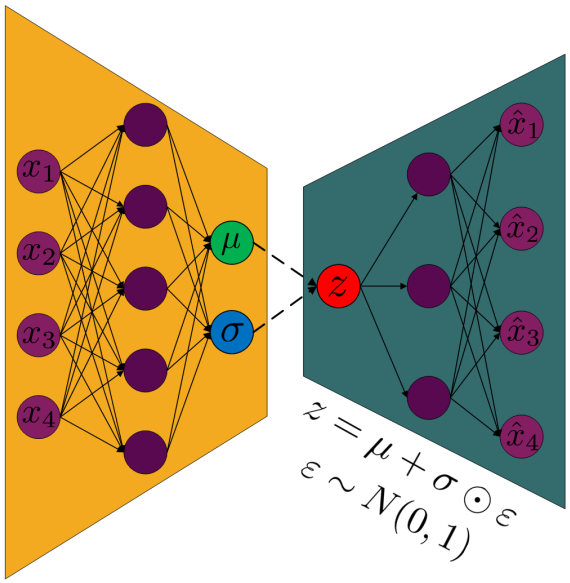


FIG. 2: An example of a variational self-encoder, in which each network consists of an input layer, a hidden layer and an output layer. The dashed arrows indicate the mean μ and variance σ for the reparametric Eq.12. Each cell in the output layer of the decoder requires the corresponding μ and σ

the random variable z as a differentiable and invertible variation g of another auxiliary random variable ϵ (i.e., $z = g(\mu, \sigma, \epsilon)$). We employ a re-referencing trick such that $g(\mu, \sigma, \epsilon) = \mu + \sigma \odot \epsilon$ i.e.

$$z = \mu + \sigma \odot \epsilon \tag{12}$$

where $\epsilon \sim N(0,1)$ and $\odot$ is the product of elements.

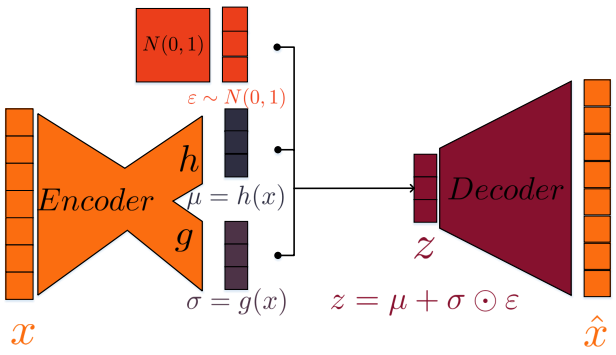


FIG. 3: Variational autoencoder, a VAE is consist of two parts, encoder and decoder, and two neural networks are used in the output stage of the encoder to obtain the mean μ and variance σ respectively. The input z to the decoder requires μ and σ obtained by the reparameterization trick [see Eq.12]

III. SPIN VARIATIONAL GRAPH AUTO-ENCODER

We want to construct a graph variational self-encoder that applies the idea of VAE to graph structure data. We want our graph variational self-encoder to generate new Ising configurations of different sizes in steady state. However, we cannot use VAE directly because traditional convolution can only be applied to Ising configurations of the same size, and different Ising configurations are equivalent to different graph structures with different numbers of nodes resulting in the inability to use convolution directly. At this point, we can use the graph convolution mentioned above, which can effectively describe different graph structures, train different sized configurations in the same neural network, and get the generation effect we want, i.e., generate steady-state Ising configurations of different sizes.

A. Encoder

The first two GCN layers generate a high-dimensional feature matrix, while we consider the edge information in the graph structure, the edge features are involved in the convolutional aggregation, refer to the previous update formula of the graph convolution can get our new update formula.

$$\mathbf{x}_i' = \mathbf{W}_1\mathbf{x}_i + \mathbf{W}_2 \sum_{j\in\mathcal{N}(i)} e_{j,i}\cdot\mathbf{x}_j \tag{13}$$

$e_{j,i}$ denotes the edge weights from source node j to target node i. The result of the upper convolution in the convolution operation is used as the input to the lower convolution to perform the stacking of convolutions.The third convolutional layer we use two different convolutional networks to obtain $\mu, \log\sigma^2$ respectively

$$\mu = GCN_\mu(\hat{X}, A) \tag{14}$$

$$\log\sigma^2 = GCN_{\log\sigma^2}(\hat{X}, A) \tag{15}$$

Then we can sample Z by eqs. 12. from the distribution.

B. Decoder

Our sampling result z is a high-dimensional representation of the nodes. In the decoder part, since the connection between the nodes of the Ising model is fixed, we can ignore the edge properties of the Ising model in the decoder part and use a fully connected neural network for each node in the potential variable z to reduce its dimensionality to 1. At this point we can think of the reconstruction problem as classifying each node, Using the sigmod activation function, we then classify them by value. We obtain our generated reconstruction model

$$\overline{x}_i = MLP(z_i) \tag{16}$$

z_i represents the high-dimensional representation of the nodes, which are transformed into the reconstructed nodes we need by the fully-connected neural network. Since we perform the fully-connected neural network for each high-dimensional node and finally downscale to 1 dimension, our model can adapt to various sizes of configurations. We realize that one model can generate multiple configurations of various sizes, Meanwhile, because of the mixed training of multiple models, some noise is added to the training, which enhances the robustness of the model and the model generation is better.

After several iterations, we can consider the model training successful when the value of the loss function is less than the threshold, and then we can save the model and the trained μ and σ and use the decoder to generate the configuration we want. Our model can generate multiple sizes of configurations, just let the sizes you need to train together, and after the training is finished, we can use the configurations under the size to fix the model to the size we need, then we can generate the configurations under the size we want.

IV. SIMULATION RESULTS AND ANALYSIS

To verify the effectiveness of the network described above, we build an Ising simulator that produces Ising states in a 32 × 32 lattice near its critical temperature. The model we are simulating is descried by a Hamiltonian of the eqs.1

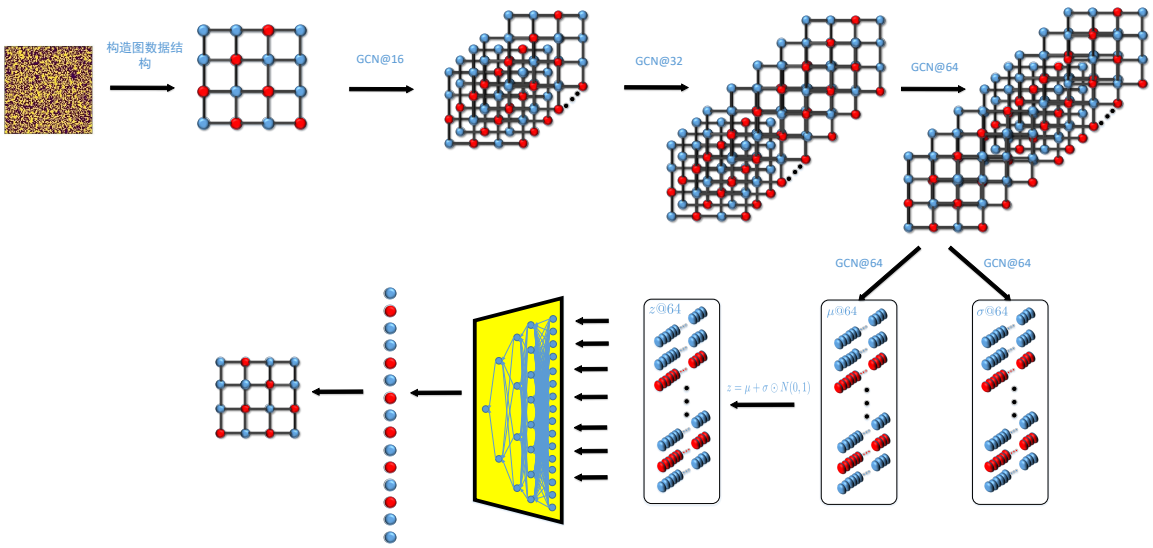
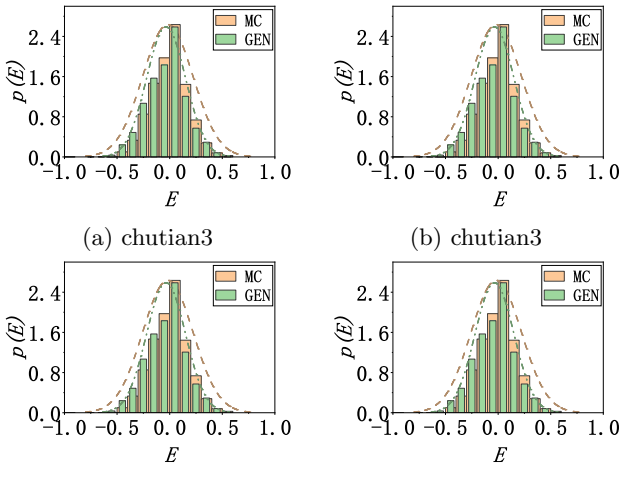


FIG. 4: Use the figure* environment to get a wide figure that spans the page in `twocolumn` formatting.



(a) chutian3 (b) chutian3

(c) chutian3 (d) chutian3

FIG. 5: mother of chutian

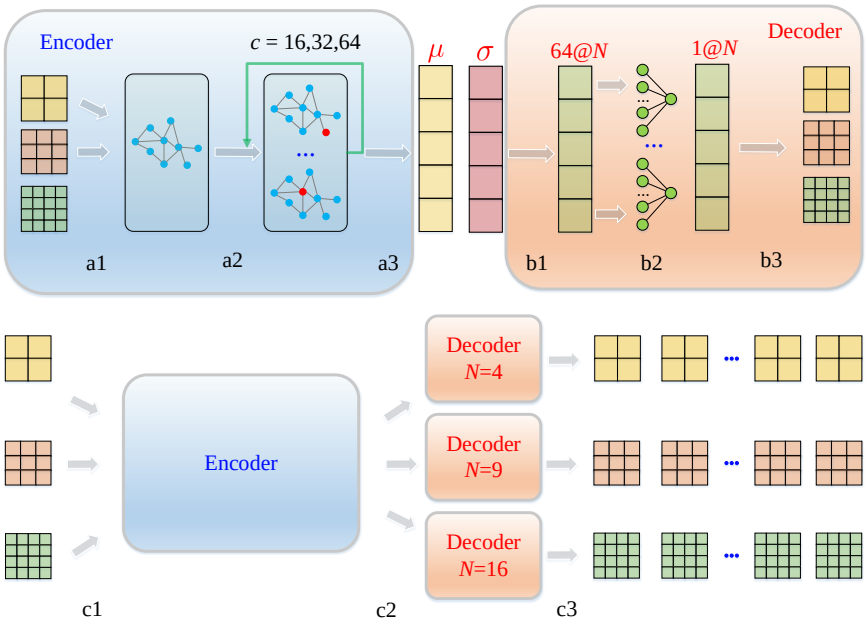


FIG. 6

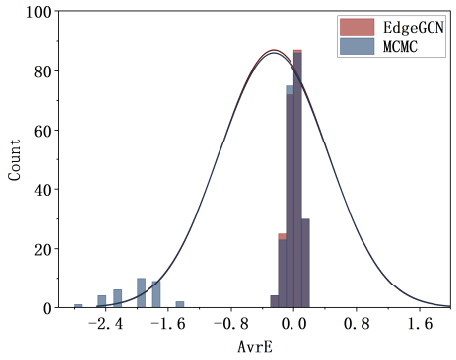


FIG. 7: Comparison of energy scatter diagrams generated by the hybrid model of 16 and 32 sizes for 32 sizes at T=3 temperature with those generated by the MCMC method

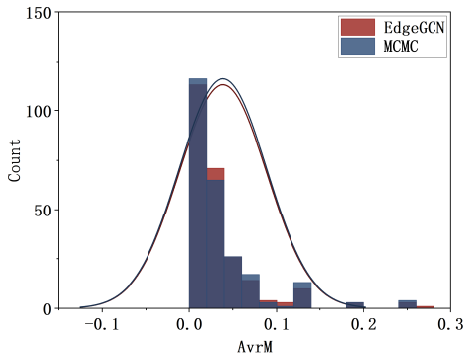


FIG. 8: Comparison of Magnetization scatter diagrams generated by a separate 32-size model at T=3 temperature with those generated by conventional methods

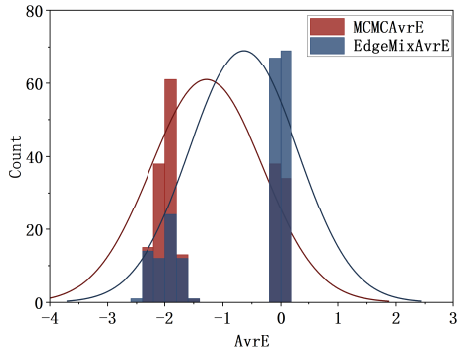


FIG. 9: Comparison of Magnetization scatter diagrams generated by the hybrid model of 16 and 32 sizes for 32 sizes at T=3 temperature with those generated by the MCMC method

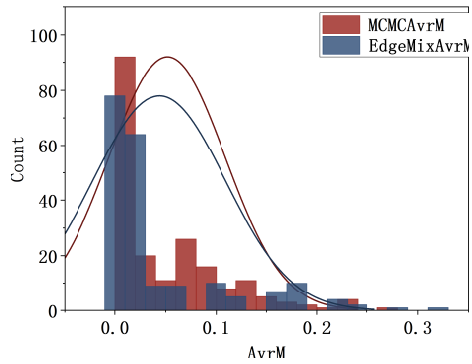


FIG. 10: Comparison of energy diagrams generated by the hybrid model of 32 and 128 sizes for 128 sizes at T=2.269 temperature with those generated by the MCMC method

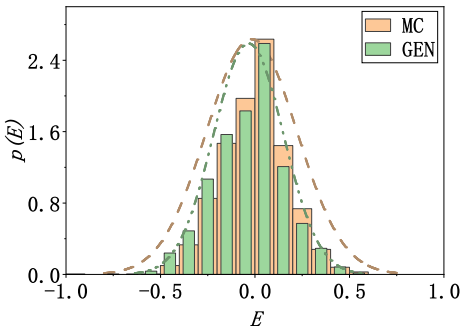


FIG. 11: omparison of energy scatter diagrams generated by the hybrid model of 16 sizes for 16 sizes at T=PTP temperature with those generated by the MCMC method

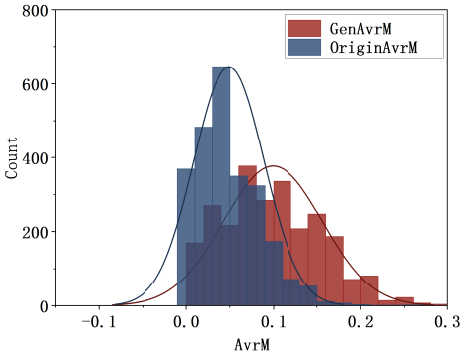


FIG. 12: omparison of energy scatter diagrams generated by the hybrid model of 16 sizes for 16 sizes at T=PTP temperature with those generated by the MCMC method