Narrow the Input Mismatch in Deep Graph Neural Network Distillation

Qiqi Zhou
Hong Kong University of Science and Technology
Hong Kong SAR, China
qzhouam@connect.ust.hk

Yanyan Shen
Shanghai Jiao Tong University
Shanghai, China
shenyy@sjtu.edu.cn

Lei Chen
Hong Kong University of Science and Technology
Hong Kong SAR, China
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
leichen@cse.ust.hk

ABSTRACT

Graph neural networks (GNNs) have been widely studied for modeling graph-structured data. Thanks to the over-parameterization and large receptive field of deep GNNs, “deep" is a promising direction to develop GNNs further and has shown some superior performances. However, the over-stacked structures of deep architectures incur high inference cost in deployment. To compress deep GNNs, we can use knowledge distillation (KD) to make shallow student GNNs mimic teacher GNNs. Existing KD methods in graph domain focus on constructing diverse supervision on embedding or prediction produced by student GNNs, but overlook the gap of the receptive field (i.e., input information) between student and teacher, which brings difficulties to KD. We call this gap “input mismatch". To alleviate this problem, we propose a lightweight stochastic extended module to provide an estimation for missing input information for student GNNs. The estimator models the distribution of missing information. Specifically, we model the missing information as an independent distribution from graph level and a conditional distribution from node level (given the condition of observable input). These two estimates are optimized using a Bayesian methodology and combined into a balanced estimate as additional input to student GNNs. To the best of our knowledge, we are the first to address the “input mismatch" problem in deep GNNs distillation. Experiments on extensive benchmarks demonstrate that our method outperforms existing KD methods for GNNs in distillation performance, which confirms that the estimations are reasonable and effective.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Mathematics of computing → Graph algorithms**.

KEYWORDS

Graph Neural Networks, Knowledge Distillation, Bayesian Optimization

ACM Reference Format:
Qiqi Zhou, Yanyan Shen, and Lei Chen. 2023. Narrow the Input Mismatch in Deep Graph Neural Network Distillation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599442

1 INTRODUCTION

Graph Neural Networks (GNNs) have recently introduced the power of Neural Networks into tasks with graph-structured data, like node classification, link prediction, and graph classification [40, 42, 47]. As a result, GNNs have shown great success in many graph-related applications, such as recommendation systems [10], fraud detection [38], biochemistry [14] and combinatorial optimization [26].

The fundamental idea of GNNs is to gather neighbor information iteratively. And L-layer GNNs are used to obtain information of L-hop neighbors for each node. Although many classical GNNs only have a few layers due to over-smoothing [5, 31] from naïvely stacking layers, there is evidence showing that over-parameterized models have better generalization ability [1, 30]. The massive success of very deep models on grid-structured data also demonstrates the appeal of deep architectures [3, 9, 33]. Besides, more layers in GNN mean a larger receptive field and more neighbor information for representing one specific node. Hence, some recent attempts [6, 24, 25] aim to overcome the over-smoothing issue brought by stacking more layers. These approaches achieve a good tradeoff between preventing over-smoothing and utilizing over-parameterization, which enable the creation of deeper GNN architectures that show superior performance. To the best of our knowledge, so far, the deep architecture GCNII [6] still ranks first on the leaderboard of *Cora*[1] and *PPI*[2] datasets. What's more, on large-scale graph benchmarks like Microsoft Graph(MAG) [39] and Open Graph Benchmark (OGB) [20], deep and large GNNs also significantly outperform shallow models, implying the great expressive power of over-parameterized GNNs. Figure 1 shows the architecture comparison on the OGB and MAG benchmarks, where the statistics are collected from public leadboard[3].

Despite the advantages of recently developed deep GNN models, their over-stacked architectures bring time-efficiency problems during inference. The reason is that the number of support nodes for the computation of one target node grows exponentially with

[1]https://paperswithcode.com/sota/node-classification-on-cora-with-public-split
[2]https://paperswithcode.com/sota/node-classification-on-ppi
[3]https://ogb.stanford.edu/docs/leader_nodeprop/

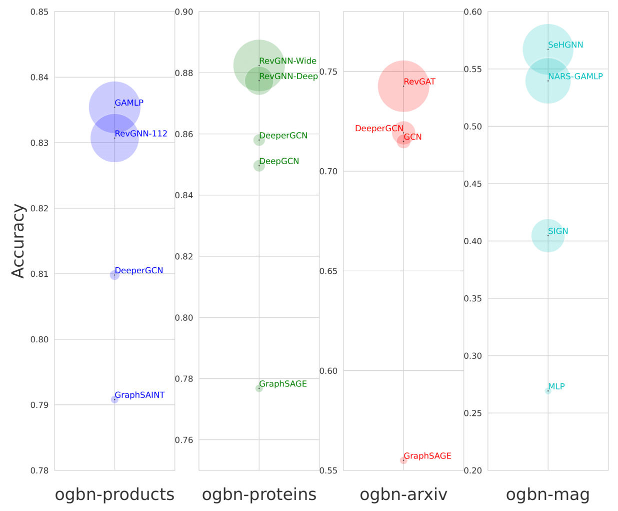


Figure 1: Node classification accuracy comparison on *ogbn-products, ogbn-proteins, ogbn-arxiv* and *ogbn-mag* benchmarks. The size of a bubble is proportional to the number of parameters of the model architecture.

L. This is the so-called "neighbor explosion" problem, which makes existing models inapplicable in time-constrained scenarios. In practice, many GNN applications are sensitive to inference delay. For example, fraud and spam detection applications [23, 28, 38] need to identify malicious transactions or entities as fast as possible to provide high-security protection for user property.

One method to alleviate this efficiency problem is Knowledge Distillation (KD). The main idea of KD follows a teacher-student architecture, and the student model imitates the behavior of the teacher model to obtain competitive performance, where the student model is smaller and faster than its teacher. KD is widely utilized for compressing huge models in visual and language tasks [15, 18, 34, 41] and is brought into GNNs recently [17, 21, 45, 48].

As shown in Figure 2, one crucial point is overlooked by existing works when distilling the knowledge of a deeper teacher GNN to a shallower student GNN. We take a 5-layer teacher model GNN_t and a 2-layer student model GNN_s as an example. As shown in Figure 2(a), when representing target node v_1, the receptive fields of GNN_s and GNN_t are subgraphs induced by nodes within 2-hops ($\mathcal{G}[V_{v_1}^n]$) and 5-hops ($\mathcal{G}[V_{v_1}^n \cup V_{v_1}^f]$), respectively, denoted as $G_{v_1}^n = \mathcal{G}[V_{v_1}^n]$ and $G_{v_1}^f = \mathcal{G}[V_{v_1}^n \cup V_{v_1}^f] - \mathcal{G}[V_{v_1}^n]$. In Figure 2(b), we compare the prediction pipelines of GNN_t and GNN_s. And all previous works' efforts can be summarized in Figure 2(b)①, constructing supervision to minimize the discrepancy of embedding space and logits between teacher and student. However, as shown in Figure 2(b)②, we can see a gap between the inputs of GNN_s and GNN_t. The missing $G_{v_1}^f$ in GNN_s can be crucial in some cases, like in molecular graphs where a molecule's chemical properties may depend on the atom combination at its opposite sides [7, 43]. As a result, this gap makes it harder for the student to approach the performance of the teacher by distillation and results in suboptimality of existing methods. We call this gap in the input domain "**input mismatch**".

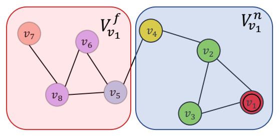
To narrow the "input mismatch" gap, one natural idea is to give an estimate of the missing $G_{v_1}^f$ for GNN_s and merge this estimation as a module into GNN_s. However, it is not trivial to estimate a $G_{v_1}^f$ serving well for GNN_s in distillation. Basically, there are three challenges. The first challenge is from the diversity of graph data. When we conduct inference with GNN_s, the only information provided for the estimation module is $G_{v_1}^n$. However, in different graph data, the magnitude of the correlation between $G_{v_1}^f$ and $G_{v_1}^n$ is diverse. Merely relying on $G_{v_1}^n$ to infer $G_{v_1}^f$ is under the assumption that there is a strong correlation between them, which is not always true. The second challenge is that inferring $G_{v_1}^f$ from $G_{v_1}^n$ is an ill-posed problem even when $G_{v_1}^f$ and $G_{v_1}^n$ are strongly correlated. That is to say, for a specific $G_{v_1}^n$, there exist multiple possible corresponding $G_{v_1}^f$. The third challenge is that the estimation module should be lightweight since one goal of GNN_s is to be lightweight. Several designs are presented to tackle these challenges in Section 4.1.

In this work, we propose a lightweight stochastic extended module, **Bayesian Receptive Field Expander** (**BRFE**), to tackle these challenges and alleviate the "input mismatch" problem. As shown in Figure 2(c), BRFE is an estimator learned from farther receptive field and teacher model, so it captures the knowledge of input graph distribution and the knowledge of the teacher model during the training phases. It can provide an embedding sampled from learned distributions for every node as an additional input to the student model, where the embedding represents an estimate for $G_{v_i}^f$ of a target node v_i. The student model is accordingly adapted to fuse the raw and additional inputs.

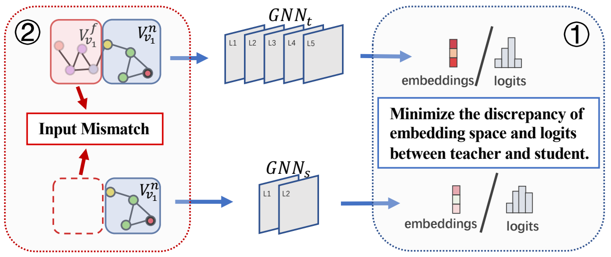
In summary, we have made the following contributions:

- To the best of our knowledge, we are the first to focus on "input mismatch" in deep GNN distillation.
- We propose BRFE, a lightweight stochastic extended module, to provide a balanced estimate of missing input for student GNN from graph-level estimation and node-level estimation, and alleviate the "input mismatch" problem.
- We evaluate our proposed BRFE by conducting extensive experiments on node classification benchmarks. Experiments demonstrate that BRFE can further improve the performance of deep GNN distillation compared to existing distillation method baselines.

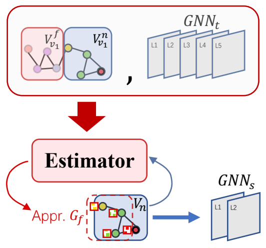
In the rest of this paper, we first give some notations and background information on GNNs and KD in Section 2. Then, a comprehensive review of previous works in the field of GNN distillation is presented in Section 3, highlighting the differences with our approach. The proposed method, BRFE, is introduced in Section 4.1. The details of the node-level and graph-level estimations are described in Section 4.2 and Section 4.3, respectively. The adaptation of student architectures and the distillation framework are presented in Section 4.4 and Section 4.5, respectively. The empirical results of BRFE are evaluated in Section 5. Finally, we conclude the paper with a discussion of the contributions and potential directions for future work in Section 6.



(a) An example graph. v_1 is the target node. A 2-layer model GNN_s represents v_1 based on receptive field $G^n_{v_1}$ (n means "near"). A 5-layer model GNN_t represents v_1 based on receptive field $G^n_{v_1}$ and $G^f_{v_1}$ (f means "far").


②
Input Mismatch
GNN_t
L1 L2 L3 L4 L5
①
embeddings
logits
Minimize the discrepancy of embedding space and logits between teacher and student.
GNN_s
L1 L2
embeddings
logits

(b) Prediction pipelines of GNN_t and GNN_s from an end-to-end view. ① focuses on minimizing the discrepancy of embedding space and logits between teacher and student. ② illustrates "input mismatch".



(c) BRFE is an estimator learned from farther receptive field and teacher model. It gives approximated G^f_v for every $v \in V_n$ as additional inputs to student GNN_s.

Figure 2: An example for illustrating "input mismatch"

2 PRELIMINARY

2.1 Notations

Let $\mathcal{G} = (V, E)$ represent the graph, where V is the set of vertices $\{v_1, \cdots, v_N\}$ with $|V| = N$ and E is the set of edges. The adjacency matrix is defined as $\boldsymbol{A} \in \{0,1\}^{N\times N}$, and $\boldsymbol{A}_{i,j} = 1$ if and only if $(v_i, v_j) \in E$. Let $\mathcal{N}(v_i) = \{v_j \mid \boldsymbol{A}_{i,j} = 1\}$ denotes the neighbors of node v_i. $\mathcal{N}^{(k)}(v_i)$ denotes the exactly k-hop neighbors of node v_i and $\mathcal{N}(v_i) = \mathcal{N}^{(1)}(v_i)$. Each node v_i is associated with a d_0-dimensional feature vector $\boldsymbol{x}_{v_i}$ and a scalar label $\bar{y}_{v_i} \in \{1, \ldots, C\}$ denoting the class of v_i. We can use $\boldsymbol{X} \in \mathbb{R}^{N\times d_0}$ (i-th row is $\boldsymbol{x}_{v_i}$) and $\bar{\boldsymbol{y}} \in \mathbb{R}^N$ (i-th element is $\bar{y}_{v_i}$) to represent all node features and labels. Our notations follows this rule[4].

2.2 Graph Neural Networks

2.2.1 GNN Layer. Most Graph Neural Networks (GNNs) follow the message-passing paradigm to generalize the convolution operation into graphs. Node v updates its representation by first aggregating the current representation of immediate neighbors $\mathcal{N}(v)$ and then combining the aggregated information and its current representation to obtain its new representation. This graph convolution procedure would be repeated multiple times, and the final representations are used as embeddings for graph tasks.

The l-th layer of the GNN message-passing scheme is:

$$\boldsymbol{m}^{(l)}_{\mathcal{N}(v)} = \text{AGGREGATE}\left(H^{(l-1)}_{\mathcal{N}(v)}\right) \tag{1a}$$

$$\boldsymbol{h}^{(l)}_{v} = \text{COMBINE}\left(\boldsymbol{m}^{(l)}_{\mathcal{N}(v)}, \boldsymbol{h}^{(l-1)}_{v}\right) \tag{1b}$$

where $H^{(l-1)}_{\mathcal{N}(v)} = \left\{\boldsymbol{h}^{(l-1)}_u : u \in \mathcal{N}(v)\right\}$, $\boldsymbol{h}^{(l)}_v$ is the representation of node v after l-th layer. Specifically, $\boldsymbol{h}^{(0)}_v$ is the original feature of node v, denoted as $\boldsymbol{x}_v$. AGGREGATE$(*)$ is the aggregation function of GNN layers, which is designed to be permutation-invariant. It aggregates a set of embedding vectors, $H^{(l-1)}_{\mathcal{N}(v)}$, to a single message vector, $\boldsymbol{m}^{(l)}_{\mathcal{N}(v)}$. COMBINE$(*)$ is the function to combine the information of neighbors and center node.

[4]In this paper, we use bold capitalized letters (e.g., $\boldsymbol{X}, \boldsymbol{H}^{(l)}$) to denote matrices; non-bold capitalized letters (e.g., $V, E, H^{(l-1)}_{\mathcal{N}(v)}$) to denote sets; bold lowercase letters (e.g., $\boldsymbol{m}^{(l)}_{\mathcal{N}(v)}, \boldsymbol{h}, \bar{\boldsymbol{y}}, \hat{\boldsymbol{y}}$) to denote vectors; non-bold lowercase letters (e.g., $\bar{y}_{v_i}, \hat{y}_{v_i}$) to denote scalars.

2.2.2 GNN Training and Testing. Here we formulate the training and testing of GNN from a probabilistic perspective. When using GNNs to handle graph learning tasks, we usually make predictions based on the embeddings of nodes (or plus a readout operation), which are produced by transforming the topology and feature of the neighbor within the receptive field. So, for simplicity, our formulation follows the node classification task.

The graph data $\mathcal{G}$ can be reformulated as the set of *(subgraph, label)* pairs $D_{(\mathfrak{r})} = \{(\mathcal{G}^{(\mathfrak{r})}_{v_i}, \bar{y}_{v_i})\}_{i=1,\ldots,N}$, where $\mathcal{G}^{(\mathfrak{r})}_{v_i}$ is the subgraph within the receptive field of center node v_i, and $\mathfrak{r}$ denotes the schema deciding the receptive field.

The training process of the model is optimizing :

$$\underset{\theta}{\text{argmin}}\, E_{(G,y)\sim\mathcal{D}_{(\mathcal{G}^{(\mathfrak{r})}_v, \bar{y}_v)}}[-\log P(y|G,\theta)] \tag{2a}$$

where $\mathcal{D}_*$ is the data distribution of $*$. In practice, the loss function of Equation (2a) is calculated by:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log P(\bar{y}_{v_i}|\mathcal{G}^{(\mathfrak{r})}_{v_i}, \theta) \tag{2b}$$

The testing process of a GNN model for node v is:

$$\hat{y}_v = GNN(\mathcal{G}^{(\mathfrak{r})}_v) = \underset{y}{\text{argmax}}\, P(y|\mathcal{G}^{(\mathfrak{r})}_v, \theta) \tag{2c}$$

where $GNN(*)$ is the model and its parameter is θ.

2.3 Knowledge Distillation

Knowledge distillation (KD) [19] is proposed to transfer the knowledge acquired by a teacher model to another smaller student model, and becomes one of the main streams of model compression and acceleration [15, 18, 34, 41]. KD methods typically construct distillation supervision by aligning the outputs, hidden feature maps, or instance relation between student and teacher.

Following our formulation, for a student GNN model $GNN_s(*)$ with parameter θ_s, the logit-based distillation supervision [19] is:

$$\mathcal{L}_d(\theta_s) = -\frac{1}{N}\sum_{i=1}^{N} \log P(\bar{y}^d_{v_i}|\mathcal{G}^{(\mathfrak{r})}_{v_i}, \theta_s) \tag{3}$$

where the distillation target labels is from the predictions of the teacher GNN, $\bar{y}^d_{v_i} = \hat{y}^t_{v_i}$, or with a temperature adjusting [19].

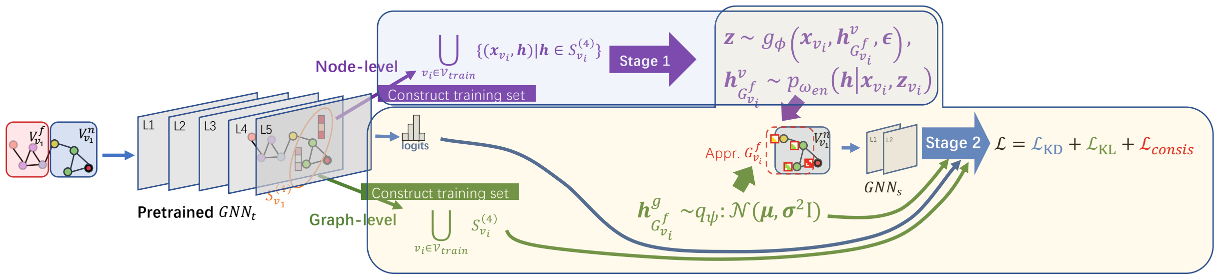

$V_{v_1}^f$
$V_{v_1}^n$
L1
L2
L3
L4
L5
$S_{v_1}^{(4)}$
Pretrained GNN_t
Node-level
Construct training set
$\bigcup_{v_i \in \mathcal{V}_{train}} \{(\boldsymbol{x}_{v_i}, \boldsymbol{h}) | \boldsymbol{h} \in S_{v_i}^{(4)}\}$
Stage 1
$\boldsymbol{z} \sim g_\phi(\boldsymbol{x}_{v_i}, \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{\epsilon})$,
$\boldsymbol{h}^v_{G^f_{v_i}} \sim p_{\omega_{en}}(\boldsymbol{h} | \boldsymbol{x}_{v_i}, \boldsymbol{z}_{v_i})$
logits
Appr. $G^f_{v_i}$
$V_{v_1}^n$
L1
L2
GNN_s
Stage 2
$\mathcal{L} = \mathcal{L}_{\mathrm{KD}} + \mathcal{L}_{\mathrm{KL}} + \mathcal{L}_{consis}$
Construct training set
Graph-level
$\bigcup_{v_i \in \mathcal{V}_{train}} S_{v_i}^{(4)}$
$\boldsymbol{h}^g_{G^f_{v_i}} \sim q_\psi : \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathrm{I})$

Figure 3: An overview of BRFE.

3 RELATED WORKS

Existing KD methods for GNNs transfer the teacher knowledge mainly by using diverse supervision constructed from soft targets, hidden feature maps, or relations to update students' transformation parameters. Yang et al. [44] proposed a novel student model combining label propagation and feature transformation, but only used a normal distillation framework with soft labels from teachers as supervision. LSP [45] takes the local structure of pairwise similarities with immediate neighbors in the teacher's node embedding space as targets for students to mimic. Joshi et al. [21] extended the local structure targets in LSP to global contrastive distillation targets in G-CRD, where they maximize the similarity among positive pairs (student and teacher feature vectors corresponding to the same node) while pushing away the feature vectors of negative pairs (unmatched nodes). GraphAKD [17] avoids using handcrafted and fixed distance functions to measure the discrepancy between teacher and student GNNs, but adversarially trains a discriminator and a student GNN (as a generator) to detect and decrease the discrepancy adaptively. GLNN [48] aggressively uses MLP as a student model. However, its distillation strategy is still simply learning knowledge from the teacher model's final predictions, and its settings require available prediction results of nodes in the testing set or plenty of observable nodes outside the training and testing sets. These works only focus on the discrepancy in embedding and logits space but overlook this gap of "input mismatch".

4 BAYESIAN RECEPTIVE FIELD EXPANDER

4.1 Overview

An overview of BRFE is shown in Figure 3. Specifically, instead of modeling the estimate of $G^f_{v_i}$ as a graph, we model it as an embedding $\boldsymbol{h}_{G^f_{v_i}}$, because we hope to keep modeling lightweight and fortunately the teacher model provides intermediate embeddings summarizing $G^f_{v_i}$, which can serve as a reference for estimation module to learn. To handle the ill-posedness challenge of estimation, we resort to Bayesian methodology [4, 8]. We model $\boldsymbol{h}_{G^f_{v_i}}$ as a random variable to highlight its uncertainty and non-uniqueness. As a result, the estimate is not optimized as a point estimate, but as a distribution. Furthermore, we represent $\boldsymbol{h}_{G^f_{v_i}}$ as a combination of two components from two different levels. BRFE models $\boldsymbol{h}_{G^f_{v_i}}$ as node-level component $\boldsymbol{h}^v_{G^f_{v_i}}$ and graph-level component $\boldsymbol{h}^g_{G^f_{v_i}}$. They correspond to two stochastic estimators to learn. First, considering the correlation between $G^f_{v_i}$ and $G^n_{v_i}$, $\boldsymbol{h}_{G^f_{v_i}}$ should have a different distribution for different $G^n_{v_i}$. We use CVAE (Conditional Variational Auto Encoder) [22, 27, 36] with a lightweight encoder to model different distributions for $G^n_{v_i}$ of different target nodes, and we represent this node-level component as $\boldsymbol{h}^v_{G^f_{v_i}}$ for node v_i. Second, as the diversity of graph data, we argue that node-level is not enough to estimate $G^f_{v_i}$ well. We also propose a graph-level modeling, which decouples $\boldsymbol{h}_{G^f_{v_i}}$ from $G^n_{v_i}$. Graph-level modeling uses a simple and lightweight Gaussian model, which is shared by different nodes and empirically works well. This graph-level component is represented as $\boldsymbol{h}^g_{G^f_{v_i}}$ for node v_i but all nodes share the same distribution. In practice, to achieve a realistic balance on the correlation between $G^f_{v_i}$ and $G^n_{v_i}$, our BRFE uses a hyperparameter to combine these two levels of estimation, or delegate student models to learn the balance automatically.

The entire process of training GNN distillation with BRFE consists of two stages. The first stage is the training of the node-level estimator, and the second stage is the joint training of the graph-level estimator and the student model. Before training, we collect the logits and intermediate embeddings of the teacher model offline.

In the first stage of training, the node-level estimator (i.e. the CVAE) is learned by using $\bigcup_{v \in \mathcal{V}_{train}} \{(\boldsymbol{x}_v, \boldsymbol{h}) | \boldsymbol{h} \in S_v^{(4)}\}$ pairs as training data when using 5-layer teacher GNN in Figure 3. The definition of $S_v^{(j)}$ will be provided in Section 4.2.

In the second stage, we optimize both the graph-level component and the student model. The loss function of the second stage can be formalized into three parts:

$$\mathcal{L} = \mathcal{L}_{KD} + \mathcal{L}_{KL} + \mathcal{L}_{consis} \tag{4}$$

where the $\mathcal{L}_{KD}$ is from the supervision of teacher's logits, and $\mathcal{L}_{KL}$ is for the training of graph-level estimator on $\bigcup_{v \in \mathcal{V}_{train}} S_v^{(4)}$. Considering that our two components are stochastic modules and are used by sampling, we add consistency loss [11, 37] $\mathcal{L}_{consis}$ as a regularization to force the inference based on the consistency of different samplings.

In the following, the teacher model is $GNN_t(*)$ with L_t layers, and the student model is $GNN_s(*)$ with L_s layers, where $L_s < L_t$. For a node v, its receptive field in $GNN_s(*)$ and $GNN_t(*)$ are different. As shown in Figure 2, the graph data $\mathcal{G}$ can be reformulated as $D_s = \{(G^n_{v_i}, \bar{y}_{v_i})\}_{i=1,\ldots,N}$ and $D_t = \{((G^n_{v_i}, G^f_{v_i}), \bar{y}_{v_i})\}_{i=1,\ldots,N}$ for student and teacher respectively. $G^n_{v_i}$ is the subgraph within L_s hops of node v_i and $G^f_{v_i}$ is the subgraph within L_t hops of node v_i but beyond L_s hops of node v_i.

4.2 Node Level Estimation

For one specific node v_i in graph $\mathcal{G}$, $G^f_{v_i}$ consists of topology and features in its farther receptive field. A simple idea is that the node-level estimator should keep the same form with the $G^f_{v_i}$, i.e. a subgraph with topology and features. However, two problems are preventing us from taking this form. First, this estimation serves for distillation, which aims to reduce the inference cost. This subgraph form is not compact enough, and much computation is needed when using the estimator. Second, the irregular structure and excessive flexibility of the subgraph form make it hard to optimize the estimator.

Thanks to the off-the-shelf trained teacher model GNN_t, it directly provides compact embeddings for $G^f_{v_i}$. We use those embeddings as target for the estimator $\boldsymbol{h}^v_{G^f_{v_i}}$ to learn. Formally, when GNN_t computing the embedding of node v_1, the computation graph can be represented as a tree as Figure 4. And we can see, for center node v_1, its representation after $(L_t - L_s)$-th or later layers starts to provide a summary of the information of $G^f_{v_1}$. We use $S^{(j)}_{v_1} = \{\boldsymbol{h}^{(j)}_u | u \in \mathcal{N}^{(L_t-j)}(v_1)\}$ to denote the set of embeddings after j-th layer summarizing the farther receptive field $G^f_{v_1}$ of node v_1, where $\mathcal{N}^{(L_t-j)}(v)$ is the $(L_t - j)$-hop neighbors of node v and $j \geq L_t - L_s$.

Remember that our goal is to infer the estimated conditional distribution of $G^f_{v_i}$ given the condition of $G^n_{v_i}$ for different node v_i. We simply use $\boldsymbol{x}_{v_i}$ to represent $G^n_{v_i}$. And we choose the distribution of $\boldsymbol{h}$ in $S^{(L_t-1)}_{v_i}$ (i.e., $j = L_t - 1 \geq L_t - L_s$) to represent the distribution of $G^f_{v_i}$ because later layers give better summary for the student to imitate the teacher. Then the pairs in $\{(\boldsymbol{x}_{v_i}, \boldsymbol{h}) | \boldsymbol{h} \in S^{(L_t-1)}_{v_i}\}$ can be seen as samples from these node-wise distributions $\mathcal{D}_{(\boldsymbol{h}^v_{G^f_{v_i}} | \boldsymbol{x}_v)}$.

To model these distributions for node-level estimation, we can use a conditional generative model. Considering that the size of $S^{(L_t-1)}_{v_i}$ is typically small for one node v, we choose variational autoencoder (CVAE) [22, 27, 36] to learn the conditional distribution of embedding in $S^{(L_t-1)}_{v_i}$ given $\boldsymbol{x}_{v_i}$, which contains the knowledge of input graph distribution and the knowledge of the teacher model. Following CVAE, there is a latent variable $\mathbf{z}$, and those variables follow:

$$\mathbf{z} \sim p_{\omega_{en}}\left(\mathbf{z} \mid \boldsymbol{x}_{v_i}\right), \boldsymbol{h}^v_{G^f_{v_i}} \sim p_{\omega_{de}}\left(\boldsymbol{h} \mid \boldsymbol{x}_{v_i}, z_{v_i}\right)$$

where ω (ω_{en} and ω_{de}) represent the parameters of the generative model (encoder and decoder)

Let ϕ denote the variational parameters in CVAE, then we have

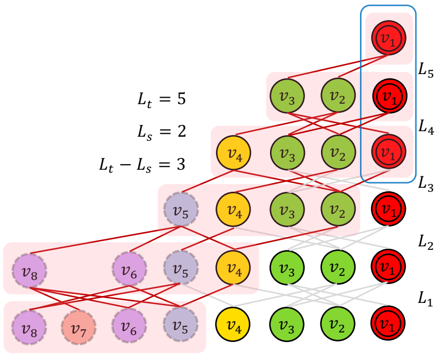


Figure 4: Computation graph of computing the embedding of v_1. Nodes circled with dash lines are in $G^f_{v_1}$. The red lines denote the spread process of information in $G^f_{v_1}$, and self-connections are omitted for clarity. The representations in red square regions contain information from $G^f_{v_1}$. The blue square circles the representation of node v_1 after $(L_t - L_s)$-th or later layers, which contain a summary of information from $G^f_{v_1}$.

$$\log p_\omega(\boldsymbol{h}^v_{G^f_{v_i}} \mid \boldsymbol{x}_{v_i}) = \int q_\phi(\mathbf{z} \mid \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{x}_{v_i}) \log \frac{p_\omega(\boldsymbol{h}^v_{G^f_{v_i}}, \mathbf{z} \mid \boldsymbol{x}_{v_i})}{q_\phi(\mathbf{z} \mid \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{x}_{v_i})} d\mathbf{z}$$
$$+ KL\left(q_\phi(\mathbf{z} \mid \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{x}_{v_i}) \| p_\omega(\mathbf{z} \mid \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{x}_{v_i})\right) \tag{5}$$

In CVAE, because of the intractable integral in Equation (5), the evidence lower bound (ELBO) [36] is used for optimization, and the loss function of CVAE can be written as:

$$\mathcal{L}_n(\boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{x}_{v_i}; \omega, \phi) = -KL\left(q_\phi(\mathbf{z} \mid \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{x}_{v_i}) \| p_{\omega_{en}}(\mathbf{z} \mid \boldsymbol{x}_{v_i})\right)$$
$$+ \frac{1}{|S^{(L_t-1)}_{v_i}|} \sum_{j=1}^{|S^{(L_t-1)}_{v_i}|} \log p_{\omega_{de}}(\boldsymbol{h}^v_{G^f_{v_i}} \mid \boldsymbol{x}_{v_i}, \mathbf{z}^{(j)}) \tag{6}$$

where $\mathbf{z}^{(j)} = g_\phi(\boldsymbol{x}_{v_i}, \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{\epsilon}^{(j)}), \boldsymbol{\epsilon}^{(j)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

$\bigcup_{v_i \in \mathcal{V}_{train}} \{(\boldsymbol{x}_{v_i}, \boldsymbol{h}) | \boldsymbol{h} \in S^{(L_t-1)}_{v_i}\}$ pairs are used to maximize the ELBO in the training stage. After training, using the decoder in CVAE, we can get the distribution of $\boldsymbol{h}^v_{G^f_{v_i}}$ given $\boldsymbol{x}_{v_i}$ by sampling a latent variable $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, as the node-level estimator. The derivations of Equation (5) and (6) are provided in Appendix A.1.

4.3 Graph Level Estimation

For graph-level estimation, we decouple $G^f_{v_i}$ from $G^n_{v_i}$, and our goal is to construct a graph-level estimator shared by nodes in graph $\mathcal{G}$. Similar to the discussion in Section 4.2, we needs to learn the distribution of embeddings in $S_g = \bigcup_{v_i \in \mathcal{V}_{train}} S^{(L_t-1)}_{v_i}$.

To model $\boldsymbol{h}^g_{G^f_{v_i}}$ with simplicity and small overhead, we use a Gaussian model, $q_\psi : \boldsymbol{h}^g_{G^f_{v_i}} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$, where ψ is the learnable parameter $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, and they are shared by all v_i. Denoting the mean vector of S_g as $\bar{\boldsymbol{h}}_g$, and variance as $\boldsymbol{\sigma}_g^2$, we use $\mathcal{N}(\bar{\boldsymbol{h}}_g, \boldsymbol{\sigma}_g^2\mathbf{I})$ as the prior distribution of $\boldsymbol{h}^g_{G^f_{v_i}}$, denoted as $p(\boldsymbol{h}^g_{G^f_u})$.

With this graph level estimator, the optimization problem according to v_i for student model GNN_s goes from Equation (2a) to Bayesian Neural Network [16] optimization problem:

$$\underset{\theta_s,\psi}{\operatorname{argmin}} - \int q_\psi(\boldsymbol{h}^g_{G^f_{v_i}}) E[\log P(y_{v_i}|G^n_{v_i}, \boldsymbol{h}^g_{G^f_{v_i}}, \theta_s)] \mathbf{d}\boldsymbol{h}^g_{G^f_{v_i}} + KL(q_\psi(\boldsymbol{h}^g_{G^f_{v_i}})||p(\boldsymbol{h}^g_{G^f_u})) \tag{7}$$

where θ_s is the parameter of GNN_s.

The KL-divergence term in Equation (7) has close-form representation:

$$KL(q_\psi(\boldsymbol{h}^g_{G^f_{v_i}})||p(\boldsymbol{h}^g_{G^f_u})) = \frac{1}{2}\left[\log\frac{|\Sigma_p|}{|\Sigma_q|} - k + (\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)^T \Sigma_p^{-1} (\boldsymbol{\mu}_q - \boldsymbol{\mu}_p) + \operatorname{tr}\left\{\Sigma_p^{-1}\Sigma_q\right\}\right] \tag{8}$$

where k is the number of dimensions of $\boldsymbol{h}^g_{G^f_{v_i}}$, $\boldsymbol{\mu}_*$ is mean vector, and Σ_* is covariance matrix.

The integral in Equation (7) can be approximated by Monte Carlo sampling as:

$$- \int q_\psi(\boldsymbol{h}^g_{G^f_{v_i}}) E[\log P(y_{v_i}|G^n_{v_i}, \boldsymbol{h}^g_{G^f_{v_i}}, \theta_s)] \mathbf{d}\boldsymbol{h}^g_{G^f_{v_i}} = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_{v_i}|G^n_{v_i}, \boldsymbol{h}^g_{G^f_{v_i}}, \theta_s) \tag{9}$$

where $\boldsymbol{h}^g_{G^f_{v_i}}$ is sampled from $q_\psi(\boldsymbol{h}^g_{G^f_u})$ and ψ is optimized by using the reparameterization trick [22].

4.4 Architecture of Student GNN with BRFE

In this section, We provide *Pre-Balance* and *Post-Balance* paradigms for balancing graph-level and node-level estimation and show two modes for injecting estimated information into student models. Then we show the whole algorithm of our distillation with BRFE.

1. Pre-Balance. In *Pre-Balance* paradigm, we balance $\boldsymbol{h}^v_{G^f_{v_i}}$ and $\boldsymbol{h}^g_{G^f_{v_i}}$ by bringing into a hyperparameter λ. The two estimates are combined into one hybrid estimate before entering the GNN model:

$$\widetilde{\boldsymbol{h}}_{v_i} = \lambda \boldsymbol{h}^v_{G^f_{v_i}} + (1-\lambda)\boldsymbol{h}^g_{G^f_{v_i}}$$

where $\boldsymbol{h}^v_{G^f_{v_i}}$ and $\boldsymbol{h}^g_{G^f_{v_i}}$ are sampled from learned node-level estimator and graph-level estimator respectively, and $\lambda \in [0, 1]$ controls how much the balance biased to node-level estimation.

The modification for injecting extra information is focused on the first layer of GNN model. Here we introduce two injecting modes, *gnn-add* and *gnn-cat*, and discuss the changes in parameter size brought by different modes.

- *gnn-add.* In *gnn-add* mode, the first layer of GNN changes to Equation (10). The number of parameter of the first layer changes from $(d_0+1)\times d_1$ to $(d_0+\widetilde{d}+2)\times d_1$, where the $\widetilde{d}$ is the dimension of $\widetilde{\boldsymbol{h}}_v$. And the output dimension of the first layer is still d_1. Typically, we have $\widetilde{d} < d_0$, so the parameter size is slightly more than the original first layer's.

$$\begin{aligned}
\overline{\boldsymbol{h}}^{(1)X}_{v_i} &= \text{COMBINE}\left(\text{AGGREGATION}\left(\left\{\boldsymbol{h}^{(0)}_u : u \in \mathcal{N}(v_i)\right\}\right), \boldsymbol{h}^{(0)}_{v_i}\right)\\
\overline{\boldsymbol{h}}^{(1)BRFE}_{v_i} &= \text{COMBINE}\left(\text{AGGREGATION}\left(\left\{\widetilde{\boldsymbol{h}}_u : u \in \mathcal{N}(v_i)\right\}\right), \widetilde{\boldsymbol{h}}_{v_i}\right)\\
\boldsymbol{h}^{(1)}_{v_i} &= \text{MEAN}(\overline{\boldsymbol{h}}^{(1)X}_{v_i}, \overline{\boldsymbol{h}}^{(1)BRFE}_{v_i})
\end{aligned} \tag{10}$$

- *gnn-cat.* In *gnn-cat* mode, the first layer of GNN changes to Equation (11), where $||$ means concatenation. We can set the dimensions of $\overline{\boldsymbol{h}}^{(1)X}_{v_i}$ and $\overline{\boldsymbol{h}}^{(1)BRFE}_{v_i}$ are both $d_1/2$. The number of parameter of the first layer changes from $(d_0+1)\times d_1$ to $(d_0+\widetilde{d}+2)\times d_1/2$, where the $\widetilde{d}$ is the dimension of $\widetilde{\boldsymbol{h}}_{v_i}$. And the output dimension of the first layer is still d_1. Typically, we have $\widetilde{d} < d_0$, so the parameter size is less than the original first layer's.

$$\overline{\boldsymbol{h}}^{(1)X}_{v_i}, \overline{\boldsymbol{h}}^{(1)BRFE}_{v_i}\text{: same as Equation (10).}$$
$$\boldsymbol{h}^{(1)}_{v_i} = \overline{\boldsymbol{h}}^{(1)X}_{v_i} || \overline{\boldsymbol{h}}^{(1)BRFE}_{v_i} \tag{11}$$

2. Post-Balance. In *Post-Balance* paradigm, we directly provide both two estimates and delegate the student model to learn the balance automatically. Similar to the discussion in *Pre-Balance*, the *gnn-add* mode and *gnn-cat* mode change the first layer as follows:

- *gnn-add.*

$$\begin{aligned}
\overline{\boldsymbol{h}}^{(1)BRFE-n}_{v_i} &= \text{COMBINE}\left(\text{AGGREGATION}\left(\left\{\boldsymbol{h}^v_{G^f_u} : u \in \mathcal{N}(v_i)\right\}\right), \boldsymbol{h}^v_{G^f_{v_i}}\right)\\
\overline{\boldsymbol{h}}^{(1)BRFE-g}_{v_i} &= \text{COMBINE}\left(\text{AGGREGATION}\left(\left\{\boldsymbol{h}^g_{G^f_u} : u \in \mathcal{N}(v_i)\right\}\right), \boldsymbol{h}^g_{G^f_{v_i}}\right)\\
\boldsymbol{h}^{(1)}_{v_i} &= \text{MEAN}(\overline{\boldsymbol{h}}^{(1)X}_{v_i}, \overline{\boldsymbol{h}}^{(1)BRFE-n}_{v_i}, \overline{\boldsymbol{h}}^{(1)BRFE-g}_{v_i}); \overline{\boldsymbol{h}}^{(1)X}_{v_i} \text{is same as Equation (10).}
\end{aligned} \tag{12}$$

The number of parameters of the first layer changes from $(d_0+1)\times d_1$ to $(d_0+2\widetilde{d}+3)\times d_1$.

- *gnn-cat.*

$$\overline{\boldsymbol{h}}^{(1)X}_{v_i}, \overline{\boldsymbol{h}}^{(1)BRFE-n}_{v_i}, \overline{\boldsymbol{h}}^{(1)BRFE-g}_{v_i}\text{: same as Equation (12).}$$
$$\boldsymbol{h}^{(1)}_{v_i} = \overline{\boldsymbol{h}}^{(1)X}_{v_i} || \overline{\boldsymbol{h}}^{(1)BRFE-n}_{v_i} || \overline{\boldsymbol{h}}^{(1)BRFE-g}_{v_i} \tag{13}$$

The number of parameters of the first layer changes from $(d_0+1)\times d_1$ to $(d_0+2\widetilde{d}+3)\times d_1/3$,

Discussion. Pre-Balance and Post-Balance differ in when the two levels of estimation are fused. Pre-Balance fuses them before entering the GNN network, which is simpler. In contrast, Post-Balance fuses them after the first GNN layer, allowing for learnable graph convolutions to process the estimations before fusion. This enhances the quality of the fused result by incorporating neighbor information and offers greater flexibility for balancing. However, Post-Balance increases the complexity of the student model. In summary, these two approaches offer different trade-offs between accuracy and efficiency, providing flexibility in selecting the best approach for the application's requirements.

4.5 GNN Distillation with BRFE

Similar to Equation (3), we have the KD supervision as:

$$\mathcal{L}_{KD} = -\frac{1}{N}\sum_{i=1}^{N}\log P(\bar{y}^d_{v_i}|G^{v_i}_n, \boldsymbol{h}^g_{G^f_{v_i}}, \boldsymbol{h}^v_{G^f_{v_i}}, \theta_s) \quad (14)$$

where θ_s is the paratmeter of GNN_s and $\bar{y}^d_{v_i}$ is the soft target from teacher model. $\boldsymbol{h}^g_{G^f_{v_i}}$ is sampled from $q_\psi(\boldsymbol{h}^g_{G^f_u})$ with trainable ψ. $\boldsymbol{h}^v_{G^f_{v_i}}$ is sampled from the trained node-level estimator(i.e. the decoder in CVAE) in the first stage.

Averaging Equation (8) of all v_i, we have $\mathcal{L}_{KL}$ still equals to Equation (8), because all v_i share the same distribution. And $\mathcal{L}_{KL}$ can be seen as a regularization on the trainable ψ in $q_\psi(\boldsymbol{h}^g_{G^f_u})$.

When using consistency loss, in every training iteration, $\boldsymbol{h}^v_{G^f_{v_i}}$, $\boldsymbol{h}^g_{G^f_{v_i}}$ sampling and forward propagation repeat K times. The consistency loss has the following form:

$$\mathcal{L}_{consis} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{N}\sum_{i=1}^{N}\left\|\overline{Z}_i - Z^{(k)}_i\right\|^2_2 \quad (15)$$

where $Z^{(k)}_i$ is the logits of instance i at k-th sampling and $\overline{Z} = \frac{1}{K}\sum_{k=1}^{K} Z^{(k)}_i$. Note that when using consistency loss, $\mathcal{L}_{KD}$ also needs to be averaged across different samplings in an iteration.

5 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the effectiveness of BRFE. Section 5.1 introduces our main experimental settings. Section 5.2 shows the overall performance of BRFE, and compares the performance of BRFR with other knowledge distillation methods. Section 5.3 analyzes the importance of the extra input provided by BRFE and Section 5.4 analyzes the balance between node-level estimation and graph-level estimation. Section 5.5 does an ablation study on our distillation method with BRFE.

5.1 Experiment Settings

Table 1: Dataset statistics.

Datasets	Classes	Nodes	Edges	Features
Cora	7	2,708	5,429	1,433
Citeseer	6	3,327	4,732	3,703
Pubmed	3	19,717	44,338	500
ogbn-arxiv	40	169,343	1,166,243	128

5.1.1 Datasets. We perform node classification on four widely-used datasets. Three small citation network datasets: *Cora*, *Citeseer*, and *Pubmed* [35]. In these datasets, nodes correspond to documents, and edges correspond to citations; each node feature corresponds to the bag-of-words representation of the document and belongs to one of the academic topics. The split schemes follow the standard fixed training/validation/testing split [46]. One large dataset from OGB benchmark [20]: *ogbn-arxiv*, representing the citation network between all Computer Science (CS) ARXIV papers; each node represents an ARXIV paper and each directed edge indicates the citation of one paper to another. The feature vector of each paper in the dataset is 128-dimensional, which is obtained by averaging the word embeddings of its title and abstract. The split scheme also follows [20]. Statistics of the datasets are summarized in Table 1. These node classification datasets all use accuracy as the evaluation metric.

5.1.2 Teacher and Student Model Settings. We choose GCNII [6], a popular deep GNN architecture, as the teacher model for *Cora*, *Citeseer*, and *Pubmed*, and these teachers follow the configuration(like the number of layers, hidden dimension and hyper-parameter, etc.) adopted for each dataset in original paper [6]. The teacher model for *ogbn-arxiv* is RevGAT [24], we use the implementation from this GitHub repository[5]. All students are two-layer GCN [40], the hidden dimension and other hyper-parameter are summarized in Appendix B.1.

5.1.3 Software and Hardware. The implementation[6] of our methods is based on the PyTorch [32], and Pytorch_geometric [12]. In addition, we conduct our experiments on NVIDIA A100-SXM4-80GB, and AMD EPYC 7413 24-Core Processor.

5.2 Overall Performance of BRFE and Comparison with Other KD Methods

To evaluate the capability of our proposed BRFE, we report node classification accuracy (mean and standard deviation of 10 runs) of distillation using BRFE on four datasets in Table 2. BRFE-g and BRFE-n mean only using graph-level and node-level estimation respectively. BRFE-gn means using both estimations. For students using BRFE, their hidden dimensions are kept the same as that of the vanilla student. Besides, the balanced paradigm and injection mode is chosen based on validation accuracy, since in practice, there are no choices consistently better in different datasets. The configuration details of these empirical results are summarized in Appendix B.1. Table 2 shows that distillation using the proposed BRFE can increase the performance of shallow student GNNs a lot and enable them to achieve comparable or even superior performance to their deep teachers. Specifically, by using distillation with BRFE, for datasets *Cora*, *Citeseer*, *Pubmed*, and *ogbn-arxiv*, the accuracy is improved by at most 2.32, 2.73, 0.27 and 2.98 on the basis of vanilla students, respectively. In addition, we can observe that BRFE-gn always performs better than BRFE-g or BRFE-n in all datasets. Table 2 also reports the time required for a single inference of the entire graph using different methods. The recorded time is the time after parallelizing the independent graph convolutions in the first layer. We can see that the architecture adaptation of the student model and estimation in BRFE do not add too much extra time cost. Because different components in the adapted first layer have no data dependency and can be easily paralleled. Distillation with BRFE can reduce up to 94.7% inference time in *ogbn-arxiv* dataset.

[5]https://github.com/lightaime/deep_gcns_torch/tree/master/examples/ogb_eff/ogbn_arxiv_dgl
[6]https://github.com/qzhou77/BRFE

Table 2: The distillation performance using BRFE on node classification benchmarks.

Method	Cora		Citeseer		PubMed		ogbn-arxiv	
	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time
Teacher	85.08 ± 0.55	7.92ms	73.6 ± 0.95	4.20ms	79.82 ± 0.59	8.31ms	73.98 ± 0.13	285ms
Vanilla Student	82.05 ± 0.56	1.31ms	71.66 ± 0.93	1.51ms	79.36 ± 0.60	2.01ms	68.69 ± 0.08	14.5ms
BRFE-g	84.18 ± 0.30(+2.13)	1.33ms	74.26 ± 0.23(+2.60)	1.57ms	80.08 ± 0.42(+0.26)	2.47ms	70.96 ± 0.18(+2.27)	15.1ms
BRFE-n	84.08 ± 0.45(+2.03)	1.49ms	73.99 ± 0.24(+2.33)	1.64ms	79.34 ± 0.33(−0.02)	2.09ms	71.33 ± 0.17(+2.64)	14.9ms
BRFE-gn	84.37 ± 0.28(+2.32)	1.44ms	74.39 ± 0.32(+2.73)	1.68ms	80.09 ± 0.29(+0.27)	2.66ms	71.67 ± 0.15(+2.98)	15.3ms

Table 3: Comparison of accuracy and parameter size between different KD methods.

Method	Cora		Citeseer		PubMed		ogbn-arxiv	
	Acc.	Size	Acc.	Size	Acc.	Size	Acc.	Size
Teacher	85.08 ± 0.55	1.35MB	73.6 ± 0.95	11.6MB	79.82 ± 0.59	16.49MB	73.98 ± 0.13	8.00MB
Vanilla Student	82.05 ± 0.56	0.70MB	71.66 ± 0.93	3.62MB	79.36 ± 0.60	0.49MB	68.69 ± 0.08	0.17MB
BRFE-g	84.01 ± 0.37	0.70MB	<u>74.13 ± 0.29</u>	3.62MB	<u>80.12 ± 0.34</u>	0.49MB	70.96 ± 0.18	0.92MB
BRFE-n	<u>84.03 ± 0.37</u>		73.94 ± 0.24		79.21 ± 0.39		<u>71.33 ± 0.17</u>	1.58MB
BRFE-gn	**84.18 ± 0.31**		**74.16 ± 0.43**		**80.15 ± 0.18**		**71.67 ± 0.15**	
KD	83.73 ± 0.46		71.83 ± 0.45		79.53 ± 0.48		69.52 ± 0.27	0.17MB
LSP	82.10 ± 0.23		69.32 ± 0.46		79.20 ± 0.55		69.25 ± 0.47	
FitNet	82.81 ± 0.58		71.95 ± 0.39		79.74 ± 0.49		69.78 ± 0.23	
GraphAKD	83.93 ± 0.86		73.33 ± 0.63		79.87 ± 0.38		68.87 ± 0.33	

To further demonstrate the superiority of BRFE, we compare it with other popular KD methods (KD [19], LSP [45], FitNet [34] and GraphAKD [17]) and report corresponding model size in Table 3. From Section 4.4, we know that the parameter size of student GNN slightly changes when using BRFE, and BRFE itself also has some parameters which should also be counted. For a fair comparison, we adjust the hidden dimensions in the student GNNs using BRFE so that all competitors have the same amount of parameters. However, the *ogbn-arxiv* dataset is an exception. Its small d_0 results in a significant impact on the model size from the adaptation of the student architecture. Therefore, we just provide the model sizes with configurations identical to those in Table 2. Despite this, the use of distillation with BRFE still reduces the model size by 81% to 88% compared to the Teacher model. All configuration details of results in Table 3 are summarized in Appendix B.1. In Table 3, the best results among different KD methods are highlighted with boldface, and the second best results are underlined. We can see that under the same model compression effect, the distillation method that uses BRFE to handle "input mismatch" consistently outperforms other distillation methods that only consider imitating the teacher model behavior. Specifically, BRFE-gn outperforms the best among existing methods by 0.25, 0.83, 0.28 and 1.89 on datasets *Cora*, *Citeseer*, *Pubmed* and *ogbn-arxiv* .

5.3 Importance Analysis of Extra Input Provided by BRFE

From Section 5.2, we see that BRFE can improve the performance of distilling knowledge from a deep GNN model into a shallow GNN model, and overall performance surpasses existing distillation methods for GNN. However, to show that BRFE indeed provides useful extra information and alleviates "input mismatch" for student GNN, we want to have a more detailed analysis of the importance of the extra input provided by BRFE in the inference of distilled student model. To analyze the importance of extra input, we use the idea from Permutation Feature Importance [2, 13, 29]. We perturb the extra input by taking a weighted sum with a random vector. Specifically, the inference of the student model can be seen as :

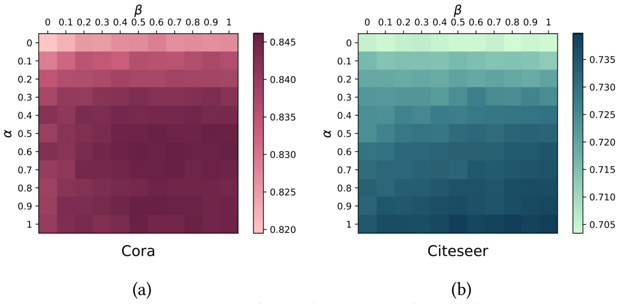


Figure 5: Importance analysis by perturbing the extra input.

$$\hat{Y} = GNN_s(G_n^v, \boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{h}^g_{G^f_{v_i}})$$

and we replace the input $\boldsymbol{h}^v_{G^f_{v_i}}$ as $\alpha \boldsymbol{h}^v_{G^f_{v_i}} + (1-\alpha)\boldsymbol{h}_{perturb-v}$, replace the input $\boldsymbol{h}^g_{G^f_{v_i}}$ as $\alpha \boldsymbol{h}^g_{G^f_{v_i}} + (1-\beta)\boldsymbol{h}_{perturb-g}$. Inference accuracy after perturbing the input in *Cora* and *Citeseer* is reported in Figure 5, where the analysis based on trained student models using *Post-Balance* paradigm and *gnn-add* mode, $\alpha, \beta \in [0, 1]$ with a step of 0.1. The $\boldsymbol{h}_{perturb-v}, \boldsymbol{h}_{perturb-g}$ vectors are first sampled from $\mathcal{N}(0, 1)$ in every dimension, and adjust the variance with the variance of $\boldsymbol{h}^v_{G^f_{v_i}}, \boldsymbol{h}^g_{G^f_{v_i}}$. The closer α and β are to 1, the more extra information from BRFE is retained.

In Figure 5(a) and (5(b)), we can observe that the color fades from the lower right corner to the upper left corner, which means that when reducing the strength of $\boldsymbol{h}_{perturb-v}$ or $\boldsymbol{h}_{perturb-g}$, the accuracy of inference drops. Besides, when α, β becomes very small, the accuracy degrades to a level close to the KD without BRFE in

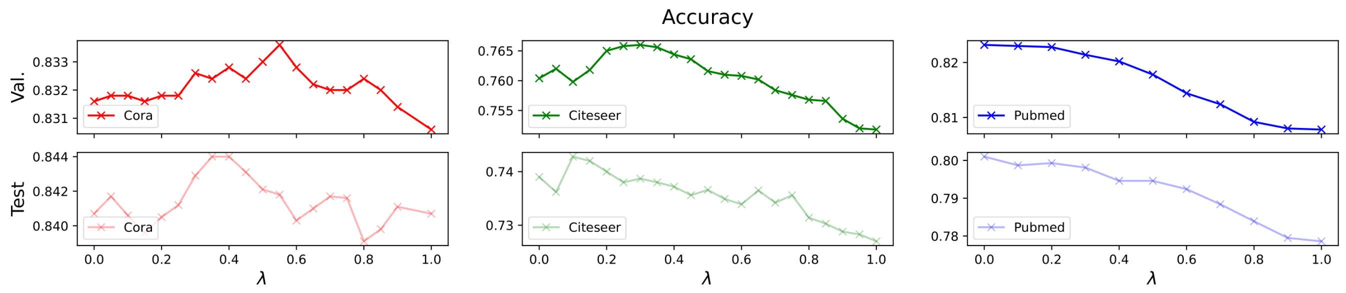


Figure 6: Balance analysis.

Table 3. As a conlusion, the extra inputs $\boldsymbol{h}^{v}_{G^{f}_{v_i}}$ and $\boldsymbol{h}^{g}_{G^{f}_{v_i}}$ provided by BRFE are critical for inference.

5.4 Balance between node-level estimation and graph-level estimation

We argue that the magnitude of the correlation between $G^{f}_{v_i}$ and $G^{n}_{v_i}$ is diverse in different graph data, and merely relying on one of node-level and graph-level estimations is suboptimal. This point is supported by the superiority of BRFE-gn in Section 3.

For a more detailed analysis of the balance between node-level and graph-level estimations in different graphs, we reported validation and test accuracy under different balancing hyper-parameter λ in *Cora*,*Citeseer* and *Pubmed* datasets. Figure 6 shows that, in all three datasets, the validation or test accuracy shows a peak at some value of λ. The validation accuracy peaks appear at $\lambda = 0.55, 0.3, 0$ for *Cora*,*Citeseer* and *Pubmed*, respectively. The test accuracy peaks appear at $\lambda = 0.4, 0.1, 0$ for *Cora*,*Citeseer* and *Pubmed*, respectively. These peaks indicate that a balanced hybrid estimation is necessary for optimal performance. What's more, the difference in the optimal λ of these peaks shows that the magnitude of the correlation between $G^{f}_{v_i}$ and $G^{n}_{v_i}$ is diverse in different graphs.

5.5 Ablation Study

To understand the contribution of every part of our whole method, we do ablation study on different components. Table 4 shows the results on *Cora*, and every column means removing some components from the complete method.

From Table 4, we have the following observations. The contribution of consistency loss is small, which means it is helpful but not critical. The combination of BRFE and KD supervision contributes significantly and is the core of our method. Removing one of BRFE-g or BRFE-n reduces the accuracy, which shows that a balanced hybrid estimation works better. When removing BRFE(-g&-n), our method degrades to the normal KD [19] method, with a noticeable drop in accuracy.

6 CONCLUSION

We propose a lightweight stochastic extended module, namely BRFE, to alleviate the "input mismatch" problem, which is critical but overlooked by existing Knowledge Distillation (KD) methods for compressing deep GNNs. BRFE provides an estimate for the missing input information in shallow student GNNs. Specifically, it models the missing information as an independent distribution from graph level and a conditional distribution from node level, and these two estimates are combined into a balanced estimate as additional input for student GNNs to improve distillation performance. Experiments on node classification benchmarks demonstrate that BRFE brings superior distillation performance than existing KD methods, which implies that the estimation is reasonable and effective. For future work, we plan to explore the potentials of BRFE in other types of graph learning tasks and large-scale GNN applications in industry.

Table 4: Ablation study.

Complete: 84.37 ± 0.28	Accuracy (%)	Δ
– Consistency Loss	84.15 ± 0.37	−0.12
– BRFE-g (preserve BRFE-n)	84.18 ± 0.30	−0.09
– BRFE-g&KD supervision	83.39 ± 0.38	−0.98
– BRFE-n (preserve BRFE-g)	84.08 ± 0.45	−0.19
– BREF-n&KD supervision	82.95 ± 0.28	−1.42
– BRFE-g&BREF-n	83.73 ± 0.46	−0.64
– BRFE-g&BREF-n&KD supervision	82.05 ± 0.56	−2.32

ACKNOWLEDGMENTS

Yanyan Shen is supported by the National Key Research and Development Program of China (2022YFE0200500), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), and SJTU Global Strategic Partnership Fund (2021SJTU-HKUST). Lei Chen is partially supported by National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16209519, CRF Project C6030-18G, C2004-21GF, AOE Project AoE/E-603/18, RIF Project R6020-19, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab grant and HKUST Global Strategic Partnership Fund (2021 SJTU-HKUST).

REFERENCES

[1] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* 116, 32 (2019), 15849–15854.
[2] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324
[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda

Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Daniela Calvetti and Erkki Somersalo. 2018. Inverse problems: From regularization to Bayesian inference. *Wiley Interdisciplinary Reviews: Computational Statistics* 10, 3 (2018), e1427.

[5] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3438–3445.

[6] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*. PMLR, 1725–1735.

[7] Tianlong Chen, Kaixiong Zhou, Keyu Duan, Wenqing Zheng, Peihao Wang, Xia Hu, and Zhangyang Wang. 2022. Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[8] Masoumeh Dashti and Andrew M Stuart. 2017. The Bayesian approach to inverse problems. In *Handbook of uncertainty quantification*. Springer, 311–428.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. 2019. Metapath-guided heterogeneous graph neural network for intent recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2478–2486.

[11] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems* 33 (2020), 22092–22103.

[12] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[13] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. 2019. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *J. Mach. Learn. Res.* 20, 177 (2019), 1–81.

[14] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. 2017. Protein interface prediction using graph convolutional networks. *Advances in neural information processing systems* 30 (2017).

[15] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born again neural networks. In *International Conference on Machine Learning*. PMLR, 1607–1616.

[16] Yarin Gal. 2016. *Uncertainty in Deep Learning*. Ph. D. Dissertation. University of Cambridge.

[17] Huarui He, Jie Wang, Zhanqiu Zhang, and Feng Wu. 2022. Compressing Deep Graph Neural Networks via Adversarial Knowledge Distillation. *arXiv preprint arXiv:2205.11678* (2022).

[18] Byeongho Heo, Jeesoo Kim, Sangdoo Yun, Hyojin Park, Nojun Kwak, and Jin Young Choi. 2019. A comprehensive overhaul of feature distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1921–1930.

[19] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).

[20] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[21] Chaitanya K Joshi, Fayao Liu, Xu Xun, Jie Lin, and Chuan-Sheng Foo. 2021. On Representation Knowledge Distillation for Graph Neural Networks. *arXiv preprint arXiv:2111.04964* (2021).

[22] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[23] Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. 2019. Spam review detection with graph convolutional networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2703–2711.

[24] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021. Training graph neural networks with 1000 layers. In *International conference on machine learning*. PMLR, 6437–6449.

[25] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9267–9276.

[26] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems* 31 (2018).

[27] Songtao Liu, Rex Ying, Hanze Dong, Lanqing Li, Tingyang Xu, Yu Rong, Peilin Zhao, Junzhou Huang, and Dinghao Wu. 2022. Local augmentation for graph neural networks. In *International Conference on Machine Learning*. PMLR, 14054–14072.

[28] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2077–2085.

[29] Christoph Molnar. 2022. Interpretable machine learning. https://christophm.github.io/interpretable-ml-book/feature-importance.html

[30] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. 2019. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BygfghAcYX

[31] Kenta Oono and Taiji Suzuki. 2020. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1ldO2EFPr

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[33] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[34] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[35] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[36] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems* 28 (2015).

[37] Vikas Verma, Kenji Kawaguchi, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz. 2019. Interpolation consistency training for semi-supervised learning. *arXiv preprint arXiv:1903.03825* (2019).

[38] Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. 2019. A semi-supervised graph attentive network for financial fraud detection. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 598–607.

[39] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.

[40] Max Welling and Thomas N Kipf. 2016. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*.

[41] Peter West, Chandra Bhagavatula, Jack Hessel, Jena Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. 2022. Symbolic Knowledge Distillation: from General Language Models to Commonsense Models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Seattle, United States, 4602–4625. https://doi.org/10.18653/v1/2022.naacl-main.341

[42] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[43] Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. 2020. Tinygnn: Learning efficient graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1848–1856.

[44] Cheng Yang, Jiawei Liu, and Chuan Shi. 2021. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *Proceedings of the Web Conference 2021*. 1227–1237.

[45] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. 2020. Distilling knowledge from graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7074–7083.

[46] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*. PMLR, 40–48.

[47] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in neural information processing systems* 31 (2018).

[48] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. 2022. Graph-less Neural Networks: Teaching Old MLPs New Tricks Via Distillation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=4p6_5HBWPCw

A EQUATION DERIVATION

A.1 CVAE

The derivations of Equation (5) and (6) are general for CVAE. For brevity, we use $\boldsymbol{x}, \boldsymbol{h}, L$ to denote $\boldsymbol{x}_{v_i}, \boldsymbol{h}^{v}_{G^f_{v_i}}, |S^{(L_t-1)}_{v_i}|$, respectively.

For Equation (5),

$$\begin{aligned}
&\log p_\omega(\boldsymbol{h} \mid \boldsymbol{x}) = \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log p_\omega(\boldsymbol{h} \mid \boldsymbol{x})\, \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \boldsymbol{x})}{p_\omega(\boldsymbol{x})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \boldsymbol{x})\, p_\omega(\boldsymbol{h}, \boldsymbol{x}, \mathrm{z})}{p_\omega(\boldsymbol{x})\, p_\omega(\boldsymbol{h}, \boldsymbol{x}, \mathrm{z})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \boldsymbol{x}, \mathrm{z})}{p_\omega(\boldsymbol{x})} \frac{1}{\frac{p_\omega(\boldsymbol{h}, \boldsymbol{x}, \mathrm{z})}{p_\omega(\boldsymbol{h}, \boldsymbol{x})}} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \mathrm{z} \mid \boldsymbol{x})}{p_\omega(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \mathrm{z} \mid \boldsymbol{x})}{p_\omega(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \frac{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \left(\log \frac{p_\omega(\boldsymbol{h}, \mathrm{z} \mid \boldsymbol{x})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} + \log \frac{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})}{p_\omega(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \right) \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \mathrm{z} \mid \boldsymbol{x})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \mathrm{dz} \\
&\quad + KL\left(q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \| p_\omega(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})\right) \\
&\geq \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \mathrm{z} \mid \boldsymbol{x})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \mathrm{dz}
\end{aligned} \tag{16}$$

For Equation (6),

$$\begin{aligned}
L_{ELBO} &= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \mathrm{z} \mid \boldsymbol{x})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h}, \boldsymbol{x}, \mathrm{z})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})\, p_\omega(\boldsymbol{x})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h} \mid \boldsymbol{x}, \mathrm{z})\, p_\omega(\boldsymbol{x}, \mathrm{z})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})\, p_\omega(\boldsymbol{x})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\boldsymbol{h} \mid \boldsymbol{x}, \mathrm{z})\, p_\omega(\mathrm{z} \mid \boldsymbol{x})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \mathrm{dz} \\
&= \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log \frac{p_\omega(\mathrm{z} \mid \boldsymbol{x})}{q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x})} \mathrm{dz} \\
&\quad + \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log p_\omega(\boldsymbol{h} \mid \boldsymbol{x}, \mathrm{z})\, \mathrm{dz} \\
&= -KL\left(q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \| p_\omega(\mathrm{z} \mid \boldsymbol{x})\right) \\
&\quad + \int q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \log p_\omega(\boldsymbol{h} \mid \boldsymbol{x}, \mathrm{z})\, \mathrm{dz} \\
&= -KL\left(q_\phi(\mathrm{z} \mid \boldsymbol{h}, \boldsymbol{x}) \| p_\omega(\mathrm{z} \mid \boldsymbol{x})\right) + \frac{1}{L} \sum_{j=1}^{L} \log p_\omega\left(\boldsymbol{h} \mid \boldsymbol{x}, \mathrm{z}^{(j)}\right)
\end{aligned} \tag{17}$$

B EXPERIMENT DETAILS

B.1 Detailed Experimental Settings

The configurations of vanilla students in Table 2 and 3 are the same, and Table 5 represents the detail.

Table 5: Configurations of vanilla students in Table 2 and 3.

Dataset	Hyperparameter
Cora	layer:2,hidden:128,dropout:0.6, opti:*Adam*,lr:0.01,weight_decay:0.0005,epoch:1000
Citeseer	layer:2,hidden:256,dropout:0.6, opti:*Adam*,lr:0.01,weight_decay:0.0005,epoch:1000
Pubmed	layer:2,hidden:256,dropout:0.6, opti:*Adam*,lr:0.01,weight_decay:0.0005,epoch:1000
ogbn-arxiv	layer:2,hidden:256,dropout:0.6, opti:*Adam*,lr:0.01,weight_decay:0.0005,epoch:500

Table 6 and 7 provide the configuration details of BRFE methods in Table 2 and 3 respectively.

Table 6: Configurations of BRFE methods in Table 2.

Dataset	Method	Hyper-parameter
Cora	BRFE-g	*gnn-add*, layer:2, hidden:128, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
	BRFE-n	*gnn-add*, layer:2, hidden:128, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:1000 sample_num:4
	BRFE-gn	*Post-Balance*, *gnn-add*, layer:2, hidden:128, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
Citeseer	BRFE-g	*gnn-add*, layer:2, hidden:256, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
	BRFE-n	*gnn-cat*, layer:2, hidden:128 $*$ 2, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:1000 sample_num:4
	BRFE-gn	*Pre-Balance*,λ:0.3, *gnn-add*, layer:2, hidden:256, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
Pubmed	BRFE-g	*gnn-add*, layer:2, hidden:256, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
	BRFE-n	*gnn-cat*, layer:2, hidden:128 $*$ 2, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:1000 sample_num:4
	BRFE-gn	*Pre-Balance*,λ:0, *gnn-add*, layer:2, hidden:256, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
ogbn-arxiv	BRFE-g	*gnn-add*, layer:2, hidden:256, dropout:0, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:500 sample_num:4
	BRFE-n	*gnn-add*, layer:2, hidden:256, dropout:0.1, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:500 sample_num:4
	BRFE-gn	*Pre-Balance*,λ:0.5, *gnn-add*, layer:2, hidden:256, dropout:0.1, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:500 sample_num:4

Table 7: Configurations of BRFE methods in Table 3.

Dataset	Method	Hyper-parameter
Cora	BRFE-g	*gnn-add*, layer:2, hidden:64, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
	BRFE-n	*gnn-add*, layer:2, hidden:54, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:1000 sample_num:4
	BRFE-gn	*Post-Balance*, *gnn-add*, layer:2, hidden:52, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
Citeseer	BRFE-g	*gnn-add*, layer:2, hidden:239, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
	BRFE-n	*gnn-cat*, layer:2, hidden:173 $*$ 2, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:1000 sample_num:4
	BRFE-gn	*Pre-Balance*,λ:0.2, *gnn-add*, layer:2, hidden:173, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
Pubmed	BRFE-g	*gnn-add*, layer:2, hidden:97, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
	BRFE-n	*gnn-cat*, layer:2, hidden:97 $*$ 2, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:1000 sample_num:4
	BRFE-gn	*Pre-Balance*,λ:0, *gnn-add*, layer:2, hidden:97, dropout:0.6, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:1000 sample_num:4
ogbn-arxiv	BRFE-g	*gnn-add*, layer:2, hidden:256, dropout:0, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:500 sample_num:4
	BRFE-n	*gnn-add*, layer:2, hidden:256, dropout:0.1, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, epoch:500 sample_num:4
	BRFE-gn	*Pre-Balance*,λ:0.5, *gnn-add*, layer:2, hidden:256, dropout:0.1, dropout2:0.9, opti:*Adam*, lr:0.01, weight_decay:0.0005, g_opti:*Adam*, g_lr:0.001, epoch:500 sample_num:4