Efficient and Scalable Implicit Graph Neural Networks with Virtual Equilibrium

Anonymous submission

Abstract

It is problematic for many GNNs to capture long-range dependencies due to the over-smoothing issue, especially on large-scale graphs. Recently, Graph Equilibrium Models (GEQs) arise as a promising solution to this problem. As implicit models, GEQs give their output as the equilibrium of a fixed-point equation, so that they inherently have global receptive fields. However, to find the equilibrium, GEQs require running costly full-batch root-finding solvers from scratch during each model update, which causes severe efficiency and scalability issues and prevents them from scaling to large graphs. To address these limitations, we propose VEQ, an efficient framework for scaling GEQs to large graphs. Instead of initializing the equilibrium from scratch, VEQ adopts persistent root-finding that continues from the latest equilibrium of mini-batch nodes and their 1-hop neighbors (dubbed Virtual Equilibrium), which can effectively accelerate and calibrate the root-finding process. With this informative prior, VEQ could reach stable equilibrium in fewer steps while capturing global receptive fields. Theoretically, we provide convergence analysis for the equilibrium and gradients of VEQ. Empirically, VEQ significantly outperforms full-batch GEQs with much less training time and memory. Despite its simplicity, VEQ achieves competitive and even superior performance to many highly engineered explicit GNNs on large-scale benchmark datasets like ogbn-arxiv and ogbn-products. VEQ shows that when we resolve the efficiency and scalability issues, GEQs are indeed favorable on large graphs due to its advantage of capturing long-range dependencies.

1 Introduction

Graph Neural Networks (GNNs) (Kipf and Welling 2016) are known to suffer from the over-smoothing issue, where more depth often results in catastrophic drop in performance (Oono and Suzuki 2019; Chen et al. 2020). Consequently, GNNs are often limited to only a few hops of neighbors, *e.g.,* 4-5. Thus, it is generally hard for GNNs to capture long-range dependencies on large-scale graphs.

A new class of GNNs, dubbed Graph Equilibrium Models (GEQs), recently rise to be a promising new solution to this problem (Bai, Kolter, and Koltun 2019; Gu et al. 2020). Instead of stacking multiple *explicit* GNN layers, GEQs adopt a *single implicit layer* in the form of a fixed-point equation. It can be shown that the solution to the fixed-point equation, namely the equilibrium $\boldsymbol{z}$, is also the output of infinite explicit layers stacked together. As a result, the implicit layer has access to infinite hops of neighbors, and GEQs could benefit from *global receptive fields within one layer*. This key property further endows GEQs with promising prospects for large-scale graphs that require long-range dependencies.

However, despite the recent progress (Gu et al. 2020; Liu et al. 2021; Park, Choo, and Park 2021), existing GEQs have only been applied to small or medium graphs, and it is hard to scale GEQs to large graphs. The critical obstacle is also on the implicit layer. First, the equilibrium state $\boldsymbol{z}$ is typically obtained via iterative root-finding algorithms like Brodyen's method (Broyden 1965), which are more time-consuming than explicit layers. Second, as solving the fixed-point equation (Eq. (4)) requires global receptive fields, every forward pass of GEQs requires access to all input features and the entire adjacency matrix. In other words, existing GEQs only accommodate full-batch training. For large-scale graphs with millions of nodes, they cannot be processed (or even stored) within one GPU, and thus existing GEQs can hardly work. A simple walk-around is to adopt mini-batch training of GEQs on *subgraphs* as in explicit GNNs (Hamilton, Ying, and Leskovec 2017), but it will 1) sacrifice the key advantage of GEQs – global receptive fields – and 2) introduce large errors to the estimated equilibrium.

The efficiency and scalability issues lead to a common question: *What is the key computation bottleneck in GEQs?* We notice that the key bottleneck lies in the iterative updates in the root-finding algorithm for solving Eq. (4). As the root-finding algorithm starts from a zero or random feature, it could take hundreds of steps to reach the equilibrium $\boldsymbol{z}$, which is much slower than explicit GNNs involving only several layers. Further, during the update, it has to visit all node features at once, which dramatically increases the memory cost. Instead, if we could find an informative initialization close to the equilibrium, we can save a lot of time and memory by reducing root-finding iterations.

Motivated by this analysis, we propose **Virtual Equilibrium Model (VEQ)**, that could resolve the efficiency and scalability issues of GEQs altogether. The core idea of VEQ is to recycle the equilibrium calculated from previous model updates, and utilize them for an informative prior of mini-batch GEQ training. On the one hand, because GEQs only

change slightly between updates, the previous equilibrium is likely close to the current one, which largely reduces the iteration number needed for root-finding. On the other hand, we notice that the full-batch update of mini-batch nodes only involves their one-hop neighbors. Thus, we further utilize previous equilibrium of one-hop neighbors to calibrate the local equilibrium update in mini-batch training. As a result, VEQ could be both efficient (with only a few root-finding steps) and scalable (with only mini-batch nodes and one-hop neighbors). Meanwhile, as the equilibrium is persistently updated along training, VEQ still benefits from *global receptive fields*. We summarize main contributions as follows:

- We analyze the limitations of existing GEQs from two aspects: training inefficiency due to root-finding solvers and equilibrium deterioration in mini-batch training.
- To address the two limitations, we propose VEQ, an efficient and scalable GEQ that enjoys both fast root-finding, small memory overhead, and global receptive fields.
- Empirically, VEQ outperforms existing GEQs by a large margin (>1.5%) on all benchmark datasets, and achieves state-of-the-art performance among explicit GNN baselines on large-scale graphs like ogbn-products.

2 GEQs and Their Limitations

In this section, we first introduce the formulation of GEQs in the context of node classification, and then discuss the efficiency and scalability issues that prevent existing GEQs from solving large-scale problems.

Notations Consider a general graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$, where $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. Let $\mathcal{N}_v$ denote the set of v's neighbors in $\mathcal{G}$. Denote the input feature matrix as $\mathbf{X} \in \mathbb{R}^{n\times d}$, and the i-th row of $\mathbf{X}$ corresponds to the feature of the i-th node. Denote the adjacency matrix as $\mathbf{A} \in \mathbb{R}^{n\times n}$, where $\mathbf{A}_{u,v} = 1$ if edge $(u, v) \in \mathcal{E}$, and $\mathbf{A}_{u,v} = 0$ otherwise. Let $\mathbf{D}$ be the diagonal degree matrix of $\mathbf{A}$. Let $\hat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ denote the symmetric normalized adjacency matrix. For a feature $\boldsymbol{z}$ defined on nodes, $\boldsymbol{z}[v]$ and $\boldsymbol{z}[\mathcal{B}]$ denote the indexed feature(s) of the node v and the set $\mathcal{B} \subseteq \mathcal{V}$, respectively. We use $\bigoplus$ to denote concatenation.

2.1 Background on Full-batch GEQs

Consider a node classification task, where each node v in the graph $\mathcal{G}$ is associated with a label y. The goal of Graph Neural Networks (GNNs) is to learn a good representation $\boldsymbol{z}$ for each node, such that y can be easily predicted, *e.g.,* via a linear head. Different from explicit GNNs that stack multiple layers, Graph Equilibrium Models (GEQs) achieve this by solving a fixed-point equation, *a.k.a.* the *implicit* layer,

$$\mathbf{Z} = \Phi(\mathbf{A}, \mathbf{Z}, \mathbf{X}, \boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\theta}$ denotes layer parameters, and the solution $\mathbf{Z} \in \mathbb{R}^{n\times p}$ is the equilibrium representations of all nodes. For instance, an implicit analogy to the explicit GCN (Kipf and Welling 2016) is the following IGNN layer (Gu et al. 2020),

$$\mathbf{Z} = \sigma(\hat{\mathbf{A}}\mathbf{Z}\mathbf{W} + b_\Omega(\mathbf{X})), \tag{2}$$

where $\mathbf{W} \in \mathbb{R}^{p\times p}$ is a weight matrix, and σ is an activation function. $b_\Omega(\mathbf{X})$ is the input injection, where b_Ω is an affine transformation parameterized by Ω. The output $\mathbf{Z}$ is equivalent to the final output of an infinite-depth explicit GCN, *i.e.,* $\mathbf{Z} = \lim_{K\to\infty} \mathbf{Z}_K$, where

$$\mathbf{Z}_k = \sigma(\hat{\mathbf{A}}\mathbf{Z}_{k-1}\mathbf{W} + b_\Omega(\mathbf{X})), \quad k = 1, \dots, K. \tag{3}$$

As a result, GEQs such as IGNN have access to infinite hops of neighbors and thus enjoying global receptive fields.

To simplify the discussion, we abuse the notation a bit and adopt the vectorized form as $\boldsymbol{z} = \Phi(\boldsymbol{z}, \boldsymbol{\theta})$ by omitting $\mathbf{A}$ and $\mathbf{X}$. According to previous works (Bai, Kolter, and Koltun 2019; Xie et al. 2022), there is an equivalence between the single and multiple layer equilibrium models. Without loss of generality, we only discuss the single-layer case in the main paper, and leave the multi-layer case to Appendix D.

Equilibrium Computation Although the equilibrium $\boldsymbol{z}$ has desirable properties as above, it is less convenient to obtain. In general cases, GEQs adopt off-the-shelf iterative root-finding algorithms, such as Anderson's method (Anderson 1965) and Broyden's method (Broyden 1965). Among them, the simplest one is perhaps the following (damped) fixed-point iteration,

$$\boldsymbol{z}_k = (1 - \eta)\boldsymbol{z}_{k-1} + \eta\Phi(\boldsymbol{z}_{k-1}, \boldsymbol{\theta}), \tag{4}$$

where $\boldsymbol{z}_0$ is usually initialized as a random or all-zero vector, and $\eta \in (0, 1]$ is the damping factor. The iterations terminate when they satisfy a pre-selected condition, such as when the relative residual is lower than a tolerance ε, *i.e.,* $\|\boldsymbol{z}_{k+1} - \boldsymbol{z}_k\| / \|\boldsymbol{z}_k\| < \varepsilon$, or when the iterations exceed a maximal threshold K.

Model Training Once we have obtained $\boldsymbol{z}$, we can calculate the classification loss and update model parameters using gradient-based optimizers. As a direct application of the implicit function theorem (Krantz and Parks 2012), differentiating through $\boldsymbol{\theta}$ on both sides of Eq. (1) gives another fixed-point equation:

$$\nabla_{\boldsymbol{\theta}}\boldsymbol{z} = \nabla_{\boldsymbol{\theta}}\boldsymbol{z}\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}, \boldsymbol{\theta}), \tag{5}$$

where $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}$ denotes the gradient of $\boldsymbol{z}$ *w.r.t.* $\boldsymbol{\theta}$. Again, we can adopt any iterative root-finding algorithms to solve the equation as we do in the forward pass, without storing any intermediate activations.

2.2 Limitations of Existing GEQs

Although GEQs show promising performances on some benchmark datasets (Gu et al. 2020; Liu et al. 2021; Park, Choo, and Park 2021), these models face two critical challenges when dealing with large-scale graphs: the training inefficiency and poor scalability, as elaborated below.

Training Inefficiency Although endowed with global receptive fields, the implicit layer significantly increases the training time. During each update of model parameters, we have to solve two fixed-point equations (Eq. (1) in the forward and Eq. (5) in the backward) from scratch, which takes a long time to converge. In Figure 1, we compare the performances and the corresponding training time of IGNNs with

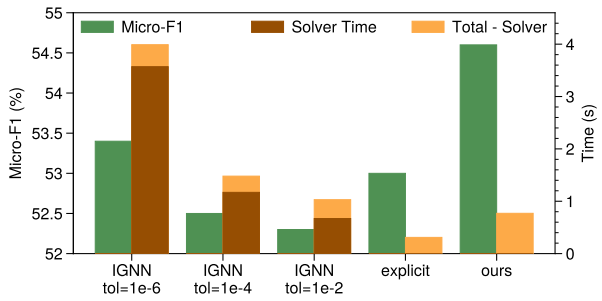

Figure 1: Micro-F1 score (%) and per-epoch training time of IGNN (Gu et al. 2020) with different tolerance ε on Flickr. We also include an explicit version of IGNN and our VEQ of similar parameter size for reference.

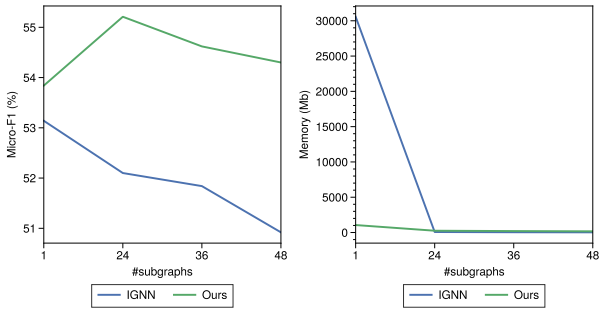

Figure 2: Micro-F1 score (left) and memory consumption (right) of IGNN (Gu et al. 2020) and our VEQ with different splits of the whole graph on Flickr.

different terminal tolerance ε. We can see that IGNN with more accurate roots (tol=10^{-6}) indeed brings better performance than explicit GNNs (53.4% *v.s.* 53.0%). However, the training is about 10 times slower, and the root-finding process indeed contributes to a major proportion (89.7%) of the total training time. Relaxing terminal tolerance to 10^{-4} or 10^{-2} can significantly reduce the training cost, but it will also severely degrade model performance with very noisy equilibrium and gradients.

Equilibrium Deterioration in Mini-batch Training Another obstacle is the scalability: existing GEQs (Gu et al. 2020; Liu et al. 2021; Park, Choo, and Park 2021) all require full adjacency matrix and all node features in their forward and backward root-finding process (Eq. (1) and Eq. (5)), which is computationally prohibitive when the graph has too many nodes[1]. Explicit GNNs usually adopt mini-batch training with different sampling techniques to scale the model to larger graphs. However, this does not suit GEQs well. As mini-batch GEQs only have access to subgraph nodes, the computed mini-batch equilibrium only has a small local receptive field, which could be quite different from the full-batch equilibrium computed with global receptive fields. As shown in Figure 2, full-batch trained GEQ is very memory-intense. And when we reduce the memory footprint by splitting the full-graph into more subgraphs, the performance also significantly degrades from 53.1% to 50.9%. Therefore, GEQs with vanilla mini-batch training suffer from equilibrium deterioration and sacrifice their global receptive fields.

[1]When the implicit layer is a linear equation, $\mathbf{Z}$ has a closed-form solution and can be directly computed (Liu et al. 2021). But as it still involves matrix inversion ($\mathcal{O}(n^3)$), the direct method is also computationally prohibitive for large-scale problem.

3 Proposed Virtual Equilibrium Model (VEQ)

Section 2 reveals that the scalability and efficiency issues of GEQs are both incurred by the high-dimensional root-finding process of full-batch training, while vanilla mini-batch training leads to equilibrium deterioration. In this section, we introduce a simple but effective approach, Virtual Equilibrium, to accelerate and scale GEQ training without performance degradation. As illustrated in Figures 1 & 2, our method is not only more efficient (faster training) and more scalable (smaller memory cost), but also superior in performance compared to full-batch GEQs.

3.1 Methodology

Equilibrium Acceleration Instead of iteratively approximating their equilibrium states from scratch with a random or zero initialization each time in the training iteration, which is very time-consuming, we propose to initialize them as their equilibrium states obtained from *previous updates*. When the model parameter is perturbed, the corresponding equilibrium will not vary too much from the original one (see Section 3.3 for a detailed explanation). Thus, our initialization strategy will significantly reduce the computation time for root-finding and accelerate GEQ training.

Equilibrium Calibration A main concern of mini-batch GEQ training, as mentioned in Section 2, is the equilibrium deterioration induced by the lack of global receptive fields. Notably, we observe that in full-batch GEQ training, updating the equilibrium of mini-batch nodes $\mathcal{B}_b$ only requires the equilibrium of their 1-hop neighbors $\mathcal{N}_{\mathcal{B}_b} := \bigcup_{v\in\mathcal{B}_b}\mathcal{N}_v$. In other words, as long as we have access to the equilibrium of the 1-hop neighbors $\mathcal{N}_{\mathcal{B}_b}$, we can obtain an accurate equilibrium of the mini-batch $\mathcal{B}_b$ without resorting to full-batch training. Motivated by this observation, we propose to include the out-of-batch 1-hop neighbors, *i.e.,* $\mathcal{N}_{\mathcal{B}_b}\backslash\mathcal{B}_b$, for computing the equilibrium of $\mathcal{B}_b$, and we adopt their equilibrium states from previous updates to approximate the exact values. These out-of-batch equilibrium states serve as reference points for computing the equilibrium of $\mathcal{B}_b$, and will *not* be updated along fixed-point iterations.

Virtual Equilibrium (VE) In summary, recycling equilibrium helps GEQ training in two aspects: previous equilibrium of $\mathcal{B}_b$ can be adopted for *accelerating root-finding*, while previous equilibrium of $\mathcal{N}_b\backslash\mathcal{B}_b$ can serve as reference points for *alleviating equilibrium bias*. To combine the benefits of both sides, we utilize the previous equilibrium of the union of both $\mathcal{B}_b$ and the 1-hop neighbors, *i.e.,* $\mathcal{V}_b := \mathcal{B}_b \cup \mathcal{N}_{\mathcal{B}_b}$, dubbed the *virtual set* of $\mathcal{B}_b$. Because the equilibrium of $\mathcal{V}_b$ contains all global information needed for com-

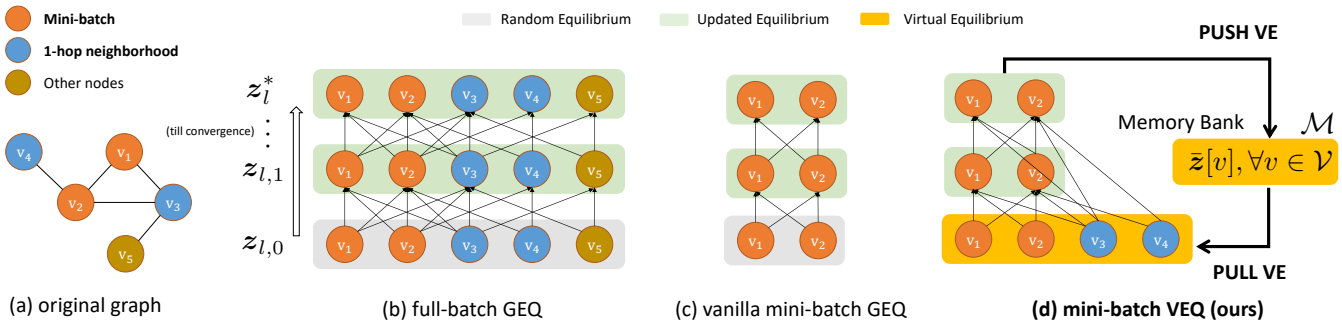

Figure 3: An illustration of the root-finding computation graph in different training strategies of GEQs. (a) An example graph. (b) Full-batch training, where the final equilibrium z_l^* has global receptive fields. (c) Vanilla mini-batch training on the subgraph of v_1, v_2, which only has local receptive field. (d) Our proposed VEQ training. We initialize v_1, v_2 with their virtual equilibrium to accelerate equilibrium computation, and initialize the 1-hop neighbors v_3, v_4 with their virtual equilibrium as reference points for updating v_1, v_2. We maintain the latest equilibrium of each node v, $\bar{z}[v]$, in a memory bank $\mathcal{M}$. As a result, the updated equilibrium of v_1, v_2 has global receptive fields in VEQ while requires much less computation compared to full-batch training.

Algorithm 1: Training of Virtual Equilibrium Model (VEQ)

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, input node features $\mathcal{X}$, number of batches B, training iterations L, the number of fixed-point iterations K.
Parameter: $\boldsymbol{\theta}_0 \in \Lambda$.

Split into mini-batches $\{\mathcal{B}_1, ..., \mathcal{B}_B\} \leftarrow \text{SPLIT}(\mathcal{G}, B)$.
Get virtual sets $\mathcal{V}_b \leftarrow \bigcup_{v \in \mathcal{B}_b} \mathcal{N}_v \cup \{v\}, \forall b \in \{1, \cdots, B\}$.
Initialize memory bank $\mathcal{M} = \{v : \bar{\boldsymbol{z}}[v], v \in \mathcal{V}\}$ (in CPU).
for $l \in \{1, ..., L\}$ **do**
  Draw a batch $\mathcal{B}_b \in \{\mathcal{B}_1, ..., \mathcal{B}_B\}$.
  Virtual initialization: $\boldsymbol{z}_{l,0}(v) \leftarrow \mathcal{M}[v], \forall v \in \mathcal{V}_b$.
  // Canonical GEQ: Initialize $\boldsymbol{z}_{l,0}[v]$ as zero, $v \in \mathcal{B}_b$.
  for k $\in \{1, ..., K\}$ **do**
    Update in-batch equilibrium: $\forall v \in \mathcal{B}_b$,

$$\begin{aligned}\boldsymbol{z}_{l,k}[v] =& \eta\Phi(\boldsymbol{z}_{l,k-1}[v \cup \mathcal{N}_v^{\text{in}}] \oplus \boldsymbol{z}_{l,0}[\mathcal{N}_v^{\text{out}}], \boldsymbol{\theta}_l) \\ &+ (1-\eta)\boldsymbol{z}_{l,k-1}[v].\end{aligned}$$

  end for
  Update memory bank $\mathcal{M}[v] \leftarrow \boldsymbol{z}_{l,K}[v], \forall v \in \mathcal{B}_b$.
  Compute $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{z}_{l,K})$ using automatic differentiation.
  Update parameters $\boldsymbol{\theta}_l \rightarrow \boldsymbol{\theta}_{l+1}$ with $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{z}_{l,K})$.
end for
return $\boldsymbol{\theta}_L$.

puting the exact equilibrium of $\mathcal{B}_b$, we can recycle the latest equilibrium of the virtual set, namely Virtual Equilibrium (VE), to accelerate and calibrate GEQs.

3.2 The Design of VEQ

Based on the methodology above, we propose Virtual Equilibrium Model (VEQ), a new scalable and efficient GEQ composed of the following components. An overview of the training process is listed in Algorithm 1.

Memory Bank We maintain a memory bank $\mathcal{M}$ for storing the latest equilibrium of all nodes in $\mathcal{V}$. We initialize them to zero vectors as in IGNN, and update the corresponding nodes after each root-finding process. If there are too many nodes in the graph, we can also store $\mathcal{M}$ in the CPU memory.

Persistent Root Finding Different from existing GEQs that initialize equilibrium from scratch, we accelerate the root-finding process by continuing from previous equilibrium. Specifically, for each mini-batch $\mathcal{B}_b$ drawn at the l-th training step, we pull the latest equilibrium of the virtual set $\mathcal{V}_b = \mathcal{B}_b \cup \mathcal{N}_b$ from $\mathcal{M}$ for *virtual initialization* as $\boldsymbol{z}_{l,0}[\mathcal{V}_b]$, and continue to update the equilibrium of in-batch nodes $\mathcal{B}_b$ with K damped fixed-point iterations. Specifically, for $v \in \mathcal{B}_b$, we split its neighbors $\mathcal{N}_v$ into two parts: the in-batch neighbors $\mathcal{N}_v^{\text{in}} = \mathcal{N}_v \cap \mathcal{B}_b$, and out-of-batch neighbors $\mathcal{N}_v^{\text{out}} = \mathcal{N}_v \backslash \mathcal{B}_b$. Then for $k = 1, \ldots, K$, we have

$$\begin{aligned}\boldsymbol{z}_{l,k}[v] =& \eta\Phi(\boldsymbol{z}_{l,k-1}[v \cup \mathcal{N}_v^{\text{in}}] \oplus \boldsymbol{z}_{l,0}[\mathcal{N}_v^{\text{out}}], \boldsymbol{\theta}_l) \\ &+ (1-\eta)\boldsymbol{z}_{l,k-1}[v].\end{aligned} \tag{6}$$

At last, we obtain the final in-batch equilibrium estimate $\boldsymbol{z}_{l,K}[\mathcal{B}_b]$ and use it to update model parameters. In practice, we notice a small K (*e.g.,* 3) is often enough for reaching a small relative error using virtual equilibrium. Afterwards, we push the new equilibrium $\boldsymbol{z}_{l,K}[\mathcal{B}_b]$ to the memory bank $\mathcal{M}$ and update the equilibrium of the mini-batch nodes $\mathcal{B}_b$.

Model Update Once we have obtained $\boldsymbol{z}_{l,K}[\mathcal{B}_b]$, we can use it to compute the node classification loss $\ell(\boldsymbol{z}_{l,K})$ and update model parameters. Specifically, following Geng et al. (2021), we utilize automatic differentiation through the K fixed-point iterations (Eq. (6)) for approximating the gradient at the equilibrium without resorting to costly implicit differentiation. Theorem 3.4 tells that the gradient is close to the exact value, and guarantees to give a descent direction of the loss function.

Discussion In Figure 3, we compare VEQ to vanilla full-batch and mini-batch training of GEQ, from which we can conclude the following major advantages of VEQ:

- **Better Efficiency.** Compared to full-batch and mini-batch training using random initialization for $\boldsymbol{z}_{l,0}$, our

virtual equilibrium initialization can be much closer to the final equilibrium, and thus largely reduce the computation cost to solve the fixed-point equation (Eq. (4)).

- **Better Scalability**. Previous GEQs all adopt full-batch training that updates all node equilibrium at each root-finding iteration. In comparison, VEQ only updates the mini-batch nodes $\mathcal{B}_b$ using 1-hop neighbors[2], which requires much less time and memory, and enables VEQ to scale to large graphs via mini-batch training.
- **Global Receptive Field.** Vanilla mini-batch GEQ training only has local receptive fields on sub-sampled graphs and thus performs much inferior to full-batch training. Instead, VEQ utilizes the virtual equilibrium of the 1-hop neighbors (which, as equilibrium states, contains knowledge of the whole graph) for updating the mini-batch equilibrium. Therefore, the mini-batch equilibrium of VEQ also has global receptive fields. In practice, we notice that VEQ is less sensitive to subgraph size and performs comparable (or even superior) to full-batch GEQs.

3.3 Theoretical Analysis

In this section, we provide theoretical analysis on the approximation of equilibrium and gradients in VEQ. For the ease of theoretical analysis, GEQs (Gu et al. 2020; Liu et al. 2021; Park, Choo, and Park 2021) generally assume the contraction of the implicit mapping, which, as Lemma 3.1 shows, guarantees the well-posedness of the fixed point $\boldsymbol{z}$, and enables the application of implicit differentiation in the backward pass. We denote $\nabla_1\Phi$ by the gradient of Φ *w.r.t.* $\boldsymbol{z}$, and $\nabla_2\Phi$ the gradient of Φ *w.r.t.* $\boldsymbol{\theta}$.

Assumption 3.1. *For every $\boldsymbol{\theta} \in \Lambda$, $\Phi(\cdot, \boldsymbol{\theta})$ is a contraction with a constant $q_{\boldsymbol{\theta}} \leq q \in (0, 1)$.*

Lemma 3.1. *Under Assumption 3.1, $\boldsymbol{z}_{\boldsymbol{\theta}}$ is the unique fixed point of $\Phi(\cdot, \boldsymbol{\theta})$, and $\mathbf{I} - \nabla_1\Phi(\boldsymbol{z}, \boldsymbol{\theta})$ is invertible.*

Equilibrium Approximation An important motivation of VEQ is the intuition that the corresponding equilibrium does not change much when the model parameter $\boldsymbol{\theta}$ is perturbed slightly. In the following, we give a theoretical description of the change on the equilibrium.

Assumption 3.2. *$\nabla_1\Phi(\boldsymbol{z}, \boldsymbol{\theta})$ and $\nabla_2\Phi(\boldsymbol{z}, \boldsymbol{\theta})$ are Lipschitz continuous for every $\boldsymbol{\theta} \in \Lambda$ with constants $L_{\Phi,\boldsymbol{z}}$ and $L_{\Phi,\boldsymbol{\theta}}$, respectively. Moreover, $\|\nabla_2\Phi(\boldsymbol{z}, \boldsymbol{\theta})\|$ is bounded by a constant $C_{\Phi,\boldsymbol{\theta}} > 0$.*

Theorem 3.2. *Under Assumptions 3.1 and 3.2, we have $\|\boldsymbol{z}_{\boldsymbol{\theta}_1} - \boldsymbol{z}_{\boldsymbol{\theta}_2}\| \leq C_1 \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|$ for $\forall\, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \Lambda$, where C_1 is a constant.*

Theorem 3.2 shows that the change of equilibrium is bounded by the change of the parameter $\boldsymbol{\theta}$. Therefore, as long as the model parameters change slightly, our virtual equilibrium recycled from previous update will be close to that of the current training step.

Another important motivation of VEQ is that if the out-of-batch VE is close to the exact value, the output of VEQ using only 1-hop VE should also be close to the exact equilibrium computed from full-batch updates. We denote $\boldsymbol{z}_{\boldsymbol{\theta}}^{in}$, $\boldsymbol{z}_{\boldsymbol{\theta}}^{out}$ as the exact node equilibrium that belongs to $\mathcal{B}$ and $\mathcal{N}[\mathcal{B}]\backslash\mathcal{B}$, respectively. We also denote $\boldsymbol{z}^{out}$ as VE that belongs to $\mathcal{N}[\mathcal{B}]\backslash\mathcal{B}$, and denote $\bar{\boldsymbol{z}}^{in}$ as the updated equilibrium that belongs to $\mathcal{B}$. The following theorem shows that the error of the updated in-batch equilibrium is bounded by the error of out-of-batch VE, which verifies our intuition.

Theorem 3.3. *Under Assumptions 3.1 and 3.2, we have $\left\|\bar{\boldsymbol{z}}^{in} - \boldsymbol{z}_{\boldsymbol{\theta}}^{in}\right\| \leq C_2 \left\|\boldsymbol{z}^{out} - \boldsymbol{z}_{\boldsymbol{\theta}}^{out}\right\|$, where C_2 is a constant.*

Gradient Approximation As described before, in the backward pass, we back-propagate through K steps of fixed-point iterations and take the gradient as an estimate of the exact value. We denote $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K$ as the estimated gradient obtained from backpropagation, and denote $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}$ as the exact gradient. With the following theorem, we prove that $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K$ converges to the exact gradient as K goes to infinity. Moreover, the error bound is related to the start of the fixed-point iteration $\boldsymbol{z}_0$. Consequently, since taking VE as the initialization narrows the gap between $\boldsymbol{z}_0$ and $\boldsymbol{z}_{\boldsymbol{\theta}}$, we can approximate the exact gradient with fewer iterations.

Theorem 3.4. *Under Assumptions 3.1 and 3.2, we have $\lim_{K\to\infty} \nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K = \nabla_{\boldsymbol{\theta}}\boldsymbol{z}$. Moreover, we have*

$$\begin{aligned}\|\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K - \nabla_{\boldsymbol{\theta}}\boldsymbol{z}\| \leq &(1 - \eta + \eta q_{\boldsymbol{\theta}})^k \|\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_0 - \nabla_{\boldsymbol{\theta}}\boldsymbol{z}\| \\ &+ k(1 - \eta + \eta q_{\boldsymbol{\theta}})^{k-1} C_3 \eta \|\boldsymbol{z}_0 - \boldsymbol{z}_{\boldsymbol{\theta}}\|,\end{aligned}$$

where C_3 is a constant.

Moreover, the following theorem guarantees that even with limited iterations, the gradient estimate still gives a descent direction of the loss landscape with mild assumptions.

Theorem 3.5. *Suppose Assumptions 3.1 and 3.2 hold, and let σ_{max} and σ_{min} be the maximal and minimal singular value of $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}$. If $\|\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K - \nabla_{\boldsymbol{\theta}}\boldsymbol{z}\| \leq \sigma_{min}^2/\sigma_{max}$, then $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K$ provides a descent direction of the loss function ℓ.*

4 Related Work

GEQs DEQs (Deep Equilibrium Models) (Bai, Kolter, and Koltun 2019; Bai, Koltun, and Kolter 2020) are a new kind of implicit models, whose output is the solution to a fixed-point equation. IGNN (Gu et al. 2020) first realizes equilibrium models on graphs and achieves superior performances than explicit GNNs at capturing long-range dependencies, though at the cost of more training time due to the root-finding process. Later works, including CGS (Park, Choo, and Park 2021) and EIGNN (Liu et al. 2021), propose to simplify the fixed-point equation to a linear form. EIGNN directly solves the closed-form solution via eigen-decomposition, which can be pre-processed for fast training. However, for a graph with n nodes and d-dimensional features, the ED step is of $\mathcal{O}(n^3)$ time complexity and $\mathcal{O}(n^2 + d^2)$ memory complexity, making them hardly scalable to large graphs. Another direction is to adopt gradient estimates to accelerate the backward pass. Similar to ours, ITD (Grazzi et al. 2020) and Phantom Gradient (Geng et al. 2021) also replace the exact gradient by unrolling-based estimates to bypass the inversion of a high-dimensional Jacobian. However, ITD initializes the equilibrium from scratch,

[2]Each virtual set $\mathcal{V}_b = \mathcal{B}_b \cup \mathcal{N}_{\mathcal{B}_b}$ is typically only a small part in the whole graph, *e.g.,* 2% in ogbn-products in average.

Type	#nodes #edges Model	89,250 899,756 Flickr	232,965 114,615,892 Reddit	716,847 13,954,819 Yelp	56,994 818,716 PPI
Explicit	GraphSAGE	50.10 ±1.3%	95.30 ±0.1%	63.40 ±0.6%	61.20
	FastGCN	50.40 ±1.0%	92.40 ±0.1%	26.50 ±5.3%	-
	VR-GCN	48.20 ±3.0%	96.40 ±0.1%	64.00 ±0.2%	85.60
	Cluster-GCN	48.10 ±0.5%	95.40 ±0.1%	60.90 ±0.5%	99.36
	GraphSAINT	51.10 ±0.1%	**97.00 ±0.1%**	**65.30 ±0.3%**	**99.50**
	GAS	53.70	95.45	62.94	98.92
	GraphFM	54.46	95.40	-	-
Implicit	IGNN	53.40 ±0.1%*	94.54 ±0.1%*	63.58 ±0.1%*	97.60
	VEQ	**55.34 ±0.7%**	96.81 ±0.3%	64.90 ±0.1%	99.22 ±0.2%

Table 1: Node classification results on large-scale graphs: micro-F1 score (%) ± standard deviation over different initialization. We mark the results implemented by us with ∗.

Type	#nodes #edges Model	169,343 1,166,243 ogbn-arxiv	2,449,029 61,859,140 ogbn-products
Explicit	GraphSAGE	71.49	78.70
	Cluster-GCN	-	78.97
	GraphSAINT	-	79.08
	GAS	71.68	76.66
	GraphFM	71.81	76.88
Implicit	IGNN	70.98*	OOM*
	VEQ	**72.82**	**79.23**

Table 2: Node classification results on large-scale OGB datasets: micro-F1 score (%). We mark the results implemented by us with ∗. OOM: out of memory.

so that it would take much more steps to approximate the exact gradient. The Phantom Gradient starts unrolling from the equilibrium. Although it can approximate $\nabla_{\boldsymbol{\theta}} \boldsymbol{z}$ much faster than ITD, it has to calculate the exact equilibrium $\boldsymbol{z}_{\boldsymbol{\theta}}$ from scratch before the unrolling starts. In comparison, we use virtual equilibrium as a warm start and the fixed-point iteration converges in a few steps, which largely saves the backward computation cost through unrolled steps, and meanwhile we do not need an extra root-solving process before the unrolling starts.

Mini-batch Training As far as we know, we are the first to train GEQs on large-scale graphs with full-batch information. Explicit GNNs also suffer from the neighbor explosion phenomenon, where the required nodes increase exponentially over layers (Hamilton, Ying, and Leskovec 2017). Various sampling-based methods have been explored to alleviate this issue, including node-wise sampling approaches (Hamilton, Ying, and Leskovec 2017; Ying et al. 2018; Chen, Zhu, and Song 2018), layer-wise sampling approaches (Chen, Ma, and Xiao 2018; Zou et al. 2019; Huang et al. 2018) and subgraph-based sampling approaches (Chiang et al. 2019; Zeng et al. 2019; Cong et al. 2020). Several recent works also study reusing the historical node embeddings from previous training steps to approximate full-batch training, such as GAS (Fey et al. 2021) and GraphFM (Yu et al. 2022). In comparison to theirs, our VEQ has two major advantages. First, L-layer explicit GNNs are fundamentally limited to a small local receptive field with L-hop neighbors (even full-batch), while as an implicit model that updates the equilibrium persistently along training, VEQ has global receptive fields over the entire graph. Second, explicit GNNs have to store and update historical embeddings of all layers ($\mathcal{O}(L)$), which could cost a lot of time and memory with deep models. Instead, in VEQ, even if the implicit layer is very complex, we still only need to store and update the output equilibrium $\boldsymbol{z}$ in the memory bank ($\mathcal{O}(1)$), which largely reduces the memory overhead and the time for data transfer.

5 Experiments

In this section, we conduct a comprehensive analysis of VEQ on large-scale benchmark datasets. We refer to Appendix C for more details on the data statistics, network architectures, and training configurations.

Datasets We study six large-scale graphs that contain thousands or millions of nodes and edges, including Flickr (Zeng et al. 2019), Yelp (Zeng et al. 2019), Reddit (Hamilton, Ying, and Leskovec 2017), PPI (Hamilton, Ying, and Leskovec 2017), and two from OGB benchmark (Hu et al. 2021): ogbn-arxiv and ogbn-products. Among them, PPI is the only multi-graph dataset.

Baselines We compare VEQ against several representative explicit and implicit models. For explicit models, we consider two node-wise sampling methods FastGCN (Chen, Ma, and Xiao 2018) and GraphSAGE (Hamilton, Ying, and Leskovec 2017), a layer-wise sampling method VR-GCN (Chen, Zhu, and Song 2018), and two subgraph-sampling methods Cluster-GCN (Chiang et al. 2019) and GraphSAINT (Zeng et al. 2019). Besides, we also include two historical-based scalable methods: GAS (Fey et al. 2021) and GraphFM (Yu et al. 2022). We report their GCN-based (Kipf and Welling 2016) variants for a fair comparison, which share the same aggregation mechanism as ours. For

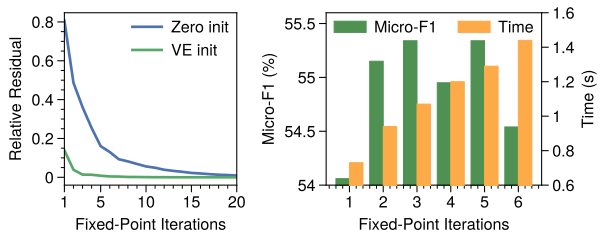

Figure 4: Left: the relative residual of root-finding with different fixed-point iterations on Flickr using zero or VE initialization. Right: micro-F1 score (%) and per-epoch training time with different fixed-point iterations on Flickr.

Training	In-batch Initialization	Out-of-batch Initialization	Micro-F1
Full Batch	Zero	-	53.68
	VE	-	53.84
Mini-Batch	Zero	-	53.20
	VE	-	53.91
	VE	VE	**55.34**

Table 3: Ablation study of different usages of virtual equilibrium (VE) on Flickr: micro-F1 score (%).

implicit models, we adopt IGNN (Gu et al. 2020) as our baseline. EIGNN (Liu et al. 2021) is not included because it barely works on large-scale datasets. For example, on the smallest Flickr dataset, its preprocessing step cannot even finish in 8 hours (10× IGNN's total training time). We do not include CGS (Park, Choo, and Park 2021) as it is only designed for graph classification tasks. For IGNN, since the authors only provide official code for PPI, we implement IGNN with the same model structure on other datasets, tune the number of layers and report the best results.

Results We repeat the experiments for 5 times on the four datasets: Flickr, Yelp, Reddit and PPI, and report the mean testing micro-F1 score and standard deviation in Table 1. We also report the results of two OGB datasets in Table 2. We can see that VEQ outperforms the implicit baseline IGNN by a large margin on the four datasets, indicating that VEQ not only enjoys global receptive fields as IGNN, but also benefits a lot from stochastic training. Besides, VEQ also achieves competitive performance against explicit methods. Specifically, it outperforms all explicit and implicit methods on the two OGB datasets, and outperforms the best explicit baseline GraphSAINT by 4.24% on Flickr. On Reddit, Yelp, and PPI, VEQ outperforms all explicit methods except GraphSAINT, while the differences to GraphSAINT are small and often not statistically significant. Notably, VEQ shows clear advantages compared to full-batch training simulators like GAS and GraphFM. This is mainly because our VEM enjoys global receptive fields with persistent root-finding along training, while an L-layer explicit model can only capture information within L-hop neighbors.

#nodes	89K	717K	233K
#edges	0.9M	14.0M	114.6M
Training	Flickr	Yelp	Reddit
Full-batch	3.2/100%	12.0/100%	6.9/100%
Mini-batch	2.3/100%	3.8/100%	3.0/100%

Table 4: Total GPU memory cost (in GB) and the proportion of data used for computing the equilibrium on Flickr.

Equilibrium Computation To verify that VE initialization indeed helps to find the equilibrium more efficiently, we show the relative residual $\left\|\boldsymbol{z}^{[i+1]}-\boldsymbol{z}^{[i]}\right\| / \left\|\boldsymbol{z}^{[i]}\right\|$ along the root-finding process in Figure 4 (left). We can see that our model exhibits stable convergence with both zero and VE initialization, while the convergence is significantly faster with VE. Indeed, as shown in Figure 4 (right), a small iteration number K is enough to attain the best performance. Even when $K = 1$, VEQ (54.1%) still outperforms IGNN (53.4%) while being considerably faster. Also, there is a tradeoff that either too many or too few iterations lead to degraded performance. $K = 3$ is an appropriate choice on Flickr, and it is still much faster than IGNN (*c.f.* Figure 1).

Ablation Study We further conduct an ablation study on the two roles of VE: 1) better initialization for in-batch nodes, and 2) reference points for out-of-batch nodes, under both full-batch and mini-batch settings. From Table 3, we can see that the in-batch VE initialization improves model performance in both cases, and the advantage is clearer under mini-batch setting (+0.71%) than full-batch setting (+0.16%). Besides, the out-of-batch virtual equilibrium is more helpful as it brings extra +1.43% micro-F1. Therefore, the global receptive fields brought by out-of-batch VE indeed contribute to a major part of the improvement of VEQ. Comparing full-batch and mini-batch training, we can see that mini-batch training performs better under VE (55.34% *v.s.* 53.84%). Meanwhile, Table 4 shows that VEQ also has less memory consumption under mini-batch training while it utilizes 100% information with global receptive fields.

6 Conclusion

In this paper, we studied the obstacles of applying existing GEQs to large-scale graphs, and found that their inefficiency and poor scalability stem from the costly root-solving process. To address these limitations, we proposed to accelerate and scale GEQ training with Virtual Equilibrium (VE). Notably, VE improves mini-batch GEQ training in two aspects: for in-batch nodes, VE serves as an informative initialization to accelerate convergence; while for out-of-batch 1-hop neighbors, VE serves as reference points that provide global information for updating in-batch nodes. Extensive experiments demonstrate that our method is both efficient and scalable, and it achieves superior performances than full-batch trained GEQs on large-scale graphs. We hope that our findings could widen the applicability of GEQs on large-scale problems, and facilitate more flexible GEQ architectures.

References

Anderson, D. G. 1965. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4): 547–560.

Bai, S.; Kolter, J. Z.; and Koltun, V. 2019. Deep Equilibrium Models. In *NeurIPS*.

Bai, S.; Koltun, V.; and Kolter, J. Z. 2020. Multiscale Deep Equilibrium Models. In *NeurIPS*.

Broyden, C. G. 1965. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92): 577–593.

Chen, D.; Lin, Y.; Li, W.; Li, P.; Zhou, J.; and Sun, X. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI Conference*, volume 34, 3438–3445.

Chen, J.; Ma, T.; and Xiao, C. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.

Chen, J.; Zhu, J.; and Song, L. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML*, 942–950. PMLR.

Chiang, W.-L.; Liu, X.; Si, S.; Li, Y.; Bengio, S.; and Hsieh, C.-J. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*, 257–266.

Cong, W.; Forsati, R.; Kandemir, M.; and Mahdavi, M. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *SIGKDD*, 1393–1403.

Fey, M.; Lenssen, J. E.; Weichert, F.; and Leskovec, J. 2021. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *ICML*, 3294–3304. PMLR.

Geng, Z.; Zhang, X.-Y.; Bai, S.; Wang, Y.; and Lin, Z. 2021. On Training Implicit Models. *NeurIPS*, 34.

Grazzi, R.; Franceschi, L.; Pontil, M.; and Salzo, S. 2020. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, 3748–3758. PMLR.

Gu, F.; Chang, H.; Zhu, W.; Sojoudi, S.; and El Ghaoui, L. 2020. Implicit Graph Neural Networks. In *NeurIPS*.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

Hu, W.; Fey, M.; Ren, H.; Nakata, M.; Dong, Y.; and Leskovec, J. 2021. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*.

Huang, W.; Zhang, T.; Rong, Y.; and Huang, J. 2018. Adaptive sampling towards fast graph representation learning. *NeurIPS*, 31.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Krantz, S. G.; and Parks, H. R. 2012. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media.

Liu, J.; Kawaguchi, K.; Hooi, B.; Wang, Y.; and Xiao, X. 2021. EIGNN: Efficient Infinite-Depth Graph Neural Networks. In *NeurIPS*.

Oono, K.; and Suzuki, T. 2019. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *ICLR*.

Park, J.; Choo, J.; and Park, J. 2021. Convergent Graph Solvers. *arXiv preprint arXiv:2106.01680*.

Xie, X.; Wang, Q.; Ling, Z.; Li, X.; Liu, G.; and Lin, Z. 2022. Optimization Induced Equilibrium Networks: An Explicit Optimization Perspective for Understanding Equilibrium Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*.

Yu, H.; Wang, L.; Wang, B.; Liu, M.; Yang, T.; and Ji, S. 2022. GraphFM: Improving Large-Scale GNN Training via Feature Momentum. In *ICML*, 25684–25701. PMLR.

Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; and Prasanna, V. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

Zou, D.; Hu, Z.; Wang, Y.; Jiang, S.; Sun, Y.; and Gu, Q. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *NeurIPS*, 32.