Efficient and Scalable Implicit Graph Neural Networks with Virtual Equilibrium

Qi Chen[1], Yifei Wang[1], Yisen Wang[2], Jianlong Chang[3], Qi Tian[3], Jiansheng Yang[1], Zhouchen Lin[2,*]
[1] *School of Mathematical Sciences, Peking University, China*
[2] *Key Lab. of Machine Perception (MoE), School of Intelligence Science and Technology, Peking University, China*
[3] *Huawei Cloud & AI, Shenzhen, China*
{chenqi77, yifei_wang, yisen.wang, jsyang, zlin}@pku.edu.cn, {jianlong.chang, tian.qi1}@huawei.com

2022 IEEE International Conference on Big Data (Big Data) | 978-1-6654-8045-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/BigData55660.2022.10020519

***Abstract*—On large-scale graphs, many graph neural networks are problematic in capturing long-range dependencies due to the oversmoothing problem. Recently, Graph Equilibrium Models (GEQs) arise as a promising solution to this issue. Their output is the equilibrium of a fixed-point equation, which can be seen as the result of iterating a GNN layer for infinite times, so that they inherently have global receptive fields. However, to find the equilibrium, GEQs require running costly full-batch root-finding algorithms from scratch during each model update, which leads to severe efficiency and scalability issues that prevent them from scaling to large graphs. To address these limitations, we propose VEQ, an efficient learning method to scale GEQs to large graphs. Instead of initializing the equilibrium from scratch in full-batch training, VEQ uses the latest equilibrium of in-batch nodes and their 1-hop neighbors (dubbed Virtual Equilibrium) to accelerate and calibrate the root-finding process in mini-batch training. With virtual equilibrium as an informative prior, VEQ is able to reach the equilibrium in fewer steps while still capturing global dependencies. Theoretically, we provide convergence analysis for the forward and backward pass of VEQ. Empirically, VEQ significantly outperforms existing GEQs by a large margin (more than 1.5%) on all benchmark datasets, with much less training time and memory. Also, 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 after we resolve the efficiency and scalability issues, GEQs are indeed favorable on large graphs due to their advantage of capturing long-range dependencies.**

***Index Terms*—graph neural networks, deep equilibrium models**

I. Introduction

Graph Neural Networks (GNNs) are powerful models for graph mining [1]–[3], facilitating applications on learning representations of graph-structured data, including social networks [4]–[7], knowledge bases [8]–[10], molecules [11]–[14], *etc.* However, they are known to suffer from the oversmoothing problem, *i.e.,* more depth often results in a catastrophic drop in performance [15]–[17]. 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.

Z. Lin was supported by the NSF China (No.s 62276004 and 61731018) and Project 2020BD006 supported by PKU-Baidu Fund. Yisen Wang was partially supported by the NSF China (No. 62006153), Project 2020BD006 supported by PKU-Baidu Fund, and Open Research Projects of Zhejiang Lab (No. 2022RC0AB05).

*Corresponding author.

A new class of GNNs, dubbed Graph Equilibrium Models (GEQs), recently rise to be a promising new solution to this problem [18]–[20]. 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, is also the output of iterating an explicit layer for infinite times. As a result, an implicit layer has access to infinite hops of neighbors, and GEQs could benefit from *global receptive fields within one layer*. This key property renders GEQs very promising when applied to large-scale graphs that require long-range dependencies.

However, despite the recent progress [18]–[20], existing GEQs have only been applied to small or medium graphs, and are difficult to scale to large graphs. We notice that the key obstacle lies in the iterative updates of the root-finding algorithm when solving the fixed-point equation. First, the equilibrium state is typically obtained via iterative root-finding algorithms like Broyden's method [21]. As the algorithms usually start from a zero or random initialization, it could take hundreds of steps to reach the equilibrium, which is much more time-consuming than explicit GNNs involving only several layers. Second, as solving the fixed-point equation 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, which dramatically increases the memory cost. 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 [22]–[24], but it will 1) sacrifice the key advantage of GEQs, *i.e.,* global receptive fields, and 2) introduce large deviations to the estimated equilibrium.

Motivated by these challenges, 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. Utilizing them as an informative prior, we can avoid finding equilibrium from scratch each time and enable mini-batch GEQ training. On the one hand, since parameters of GEQs only change slightly between model updates, the previous equilibrium is 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 1-hop neighbors. Thus, we further utilize previous equilibrium of these 1-hop neighbors to provide global information and calibrate the local equilibrium update in mini-batch training. As a result, VEQ could be both efficient (with only a few root-finding iterations) and scalable (with only mini-batch nodes and their 1-hop neighbors). Meanwhile, as the equilibrium is persistently updated along the whole training process, 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 consumption 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.

II. 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 $|\mathcal{E}| = m$. Denote the set of v's neighbors in $\mathcal{G}$ as $\mathcal{N}_v$. Denote the input feature matrix as $\mathbf{X} \in \mathbb{R}^{n\times d}$, where the i-th row 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. Let $\otimes$ denote the Kronecker product, and let $\oplus$ denote the concatenation.

A. 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 the label 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, as follows:

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

Here $\boldsymbol{\theta}$ 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 [4] is the following IGNN layer [18],

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

where $\mathbf{W}, \mathbf{U} \in \mathbb{R}^{p\times p}$ are weight matrices, and σ is an activation function. The output $\mathbf{Z}$ is equivalent to the final output of an infinite-depth explicit GNN, *i.e.,* $\mathbf{Z} = \lim_{K\to\infty} \mathbf{Z}_K$, where

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

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

To simplify the discussion, we abuse the notation a bit and adopt the vectorized form as follows by omitting $\mathbf{A}$ and $\mathbf{X}$:

$$\boldsymbol{z} = \Phi(\boldsymbol{z}, \boldsymbol{\theta}). \tag{4}$$

For example, the IGNN layer Eq. (2) can be derived in the following vectorized form:

$$\boldsymbol{z} = \sigma(\mathbf{W}^\top \otimes \hat{\mathbf{A}}\boldsymbol{z} + \mathbf{U}^\top \otimes \mathbf{I}\boldsymbol{x}), \tag{5}$$

where $\boldsymbol{z} = \mathrm{vec}(\mathbf{Z})$, $\boldsymbol{x} = \mathrm{vec}(\mathbf{X})$, and $\mathbf{I}$ denotes the identity matrix.

According to previous works [25], [26], there is an equivalence between the single and multiple layer equilibrium models. Without loss of generality, we discuss the single-layer case in the following, and leave the multi-layer case to Section III-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 [27] and Broyden's method [21]. Among them, the simplest one is perhaps the following (damped) fixed-point iteration,

$$\boldsymbol{z}_k = (1-\eta)\boldsymbol{z}_{k-1} + \eta\Phi(\boldsymbol{z}_{k-1}, \boldsymbol{\theta}), \tag{6}$$

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 a pre-selected condition is satisfied, such as when the relative residual is less than a tolerance ε, *i.e.,* $\|\boldsymbol{z}_{k+1} - \boldsymbol{z}_k\| / \|\boldsymbol{z}_k\| < \varepsilon$, or when the iteration number exceeds 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 [28], differentiating through $\boldsymbol{\theta}$ on both sides of Eq. (1) gives another fixed-point equation:

$$\nabla_{\boldsymbol{\theta}}\boldsymbol{z} = \nabla_{\boldsymbol{\theta}}\boldsymbol{z}\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}, \boldsymbol{\theta}), \tag{7}$$

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

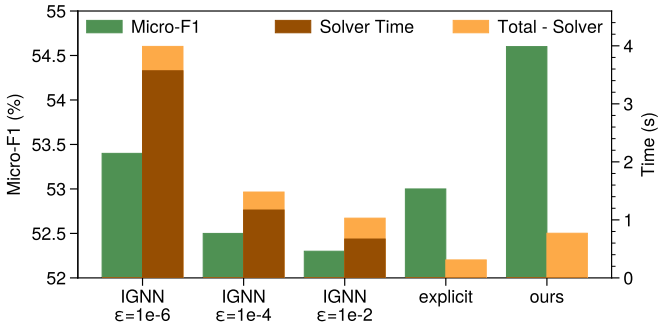

Fig. 1. Micro-F1 score and per-epoch training time of IGNN [18] with different tolerance ε on Flickr. We also include an explicit version of IGNN and our VEQ of comparable parameter size for comparison.

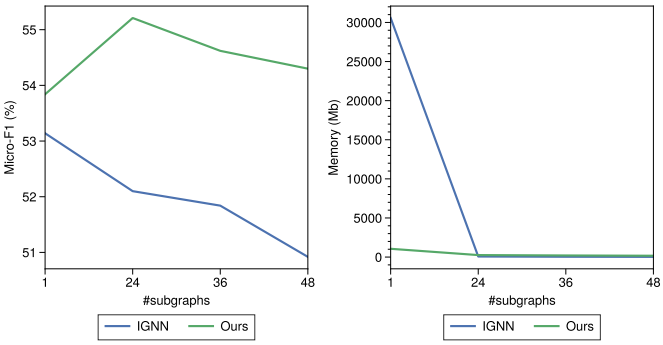

Fig. 2. Micro-F1 score (left) and memory consumption (right) of IGNN [18] and our VEQ with different numbers of subgraphs on Flickr.

B. Limitations of Existing GEQs

Although GEQs show promising performances on some benchmark datasets [18]–[20], they 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. (4) in the forward pass and Eq. (7) in the backward pass) from scratch, which takes a long time to converge. In Fig. 1, we compare the performance and the corresponding training time of IGNNs with different terminal tolerance ε. We can see that IGNN with a more accurate root ($\varepsilon = 10^{-6}$) 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 the terminal tolerance to 10^{-4} or 10^{-2} can significantly reduce the training cost, but it will also severely degrade model performance with noisy equilibrium and gradients.

Equilibrium Deterioration in Mini-batch Training Another obstacle is the scalability: existing GEQs [18]–[20] require entire adjacency matrix and all node features in their forward and backward root-finding process (Eq. (4) and Eq. (7)), which is computationally prohibitive when the graph has too many nodes[1]. Explicit GNNs usually adopt mini-batch training with sampling techniques to scale the model to larger graphs. However, this does not suit GEQs well. Different from explicit GNNs, each node equilibrium depends on global receptive fields. 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 Fig. 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%. Overall, GEQs with vanilla mini-batch training suffer from equilibrium deterioration and sacrifice their global receptive fields. Therefore, scaling GEQs using vanilla mini-batch training is not a good choice.

III. Proposed Virtual Equilibrium Model (VEQ)

Section II 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 Fig. 1 & 2, our method is not only more efficient (faster training) and scalable (smaller memory cost), but also superior in performance compared with full-batch GEQs.

A. Methodology

Equilibrium Acceleration Instead of iteratively approximating the equilibrium states from scratch in each model update, which is very time-consuming, we propose to initialize them as the 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 III-C for a detailed explanation). Thus, our initialization strategy will significantly reduce the computation time for root-finding and accelerate GEQ training.

Equilibrium Calibration Instead of considering only mini-batch information $\mathcal{B}_b$, which results in equilibrium deterioration by narrowing the receptive fields, we propose to adopt the equilibrium states of $\mathcal{N}_{\mathcal{B}_b} := \bigcup_{v\in\mathcal{B}_b} \mathcal{N}_v$ from previous model updates to approximate the equilibrium states of each mini-batch $\mathcal{B}_b$. Notably, we observe that even in full-batch GEQ training, the equilibrium of a single node can be accurately restored by the equilibrium of its 1-hop neighbors through one step of fixed-point iteration. Also, the equilibrium of mini-batch nodes $\mathcal{B}_b$ can be calculated by the equilibrium of their 1-hop neighbors $\mathcal{N}_{\mathcal{B}_b} := \bigcup_{v\in\mathcal{B}_b} \mathcal{N}_v$. Motivated by the observation, we take out-of-batch equilibrium states as

[1]When the implicit layer degenerates to a linear equation, $\mathbf{Z}$ has a closed-form solution and can be directly solved [19]. However, as it involves matrix inversion ($\mathcal{O}(n^3)$), the direct method is still computationally prohibitive for large-scale problems.

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, number of training iterations L, 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 $\mathcal{V}_b \leftarrow \mathcal{B}_b \cup \mathcal{N}_{\mathcal{B}_b}$, $\forall b \in \{1, \cdots, B\}$.
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 $\boldsymbol{\theta}_L$.

reference points to compute the equilibrium of $\mathcal{B}_b$, without resorting to full-batch computation. As these reference points approximate the exact equilibrium states, the calculated in-batch equilibrium will also approximate the accurate one (see Section III-C for a detailed explanation).

Virtual Equilibrium (VE) In summary, the equilibrium from previous model updates contributes to the mini-batch training of GEQs in two aspects: previous in-batch equilibrium ($\mathcal{B}_b$) can be adopted to *accelerate root-finding*, while previous out-of-batch equilibrium ($\mathcal{N}_{\mathcal{B}_b} \backslash \mathcal{B}_b$) can serve as reference points to *alleviate equilibrium bias*. To combine the benefits of both sides, we utilize the union of both $\mathcal{B}_b$ and their 1-hop neighbors, *i.e.,* $\mathcal{V}_b := \mathcal{B}_b \cup \mathcal{N}_{\mathcal{B}_b}$, dubbed the *virtual set* of $\mathcal{B}_b$ for mini-batch training. Because the equilibrium of $\mathcal{V}_b$ contains all global information needed to compute the exact equilibrium of $\mathcal{B}_b$, by recycling the latest equilibrium of the virtual set, namely Virtual Equilibrium (VE), we can accelerate and calibrate the current root-finding process.

B. The Design of VEQ

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

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

Persistent Root Finding In each model update, we accelerate the root-finding process by continuing from the latest equilibrium. For each mini-batch $\mathcal{B}_b$ drawn at the l-th training iteration, we pull the latest equilibrium of the virtual set $\mathcal{V}_b = \mathcal{B}_b \cup \mathcal{N}_{\mathcal{B}_b}$ from the memory bank $\mathcal{M}$ for *virtual initialization*, and continue to update the equilibrium of in-batch nodes $\mathcal{B}_b$ with K damped fixed-point iterations. Specifically, for each node $v \in \mathcal{B}_b$, 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

$$\begin{aligned}\boldsymbol{z}_{l,k}[v] =& \eta\Phi(\boldsymbol{z}_{l,k-1}[v \cup \mathcal{N}_v^{\text{in}}] \oplus \boldsymbol{z}_{l,0}[\mathcal{N}_v^{\text{out}}], \boldsymbol{\theta}_l) \\ &+ (1-\eta)\boldsymbol{z}_{l,k-1}[v],\end{aligned} \tag{8}$$

where $\boldsymbol{z}_{l,0}[\mathcal{V}_b]$ denotes the virtual initialization pulled from the memory bank. After K iterations, 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.,* $3-5$ as shown in Fig. 4) is often enough for training. Afterwards, we push the new equilibrium $\boldsymbol{z}_{l,K}[\mathcal{B}_b]$ to the memory bank $\mathcal{M}$ to update the equilibrium of the mini-batch nodes $\mathcal{B}_b$. Notably, the out-of-batch neighbors $\mathcal{N}_v^{\text{out}}$ are only pulled once and do *not* need to be updated along fixed-point iterations.

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 by gradient-based optimizers. Specifically, we utilize automatic differentiation through the K fixed-point iterations (Eq. (8)) [29] to approximate the gradient at the equilibrium without resorting to costly implicit differentiation. Theorem III.4 tells that the gradient estimate is close to the exact value, and guarantees to give a descent direction of the loss function with mild assumptions.

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

- **Better Efficiency.** Compared with random or zero initialization for $\boldsymbol{z}_{l,0}$, our *virtual equilibrium initialization* is much closer to the final equilibrium, and thus largely reduces the computation cost to solve the fixed-point equation.
- **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 their 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 sampled subgraphs, and thus performs much inferior to its full-batch trained variant. Instead, VEQ utilizes the virtual equilibrium, which contains information from the whole graph, to update the mini-batch equilibrium. Therefore, the mini-batch equilibrium of VEQ also has global receptive fields. In practice, we also notice that VEQ is less sensitive to subgraph numbers and performs comparable (or even superior) to full-batch GEQs, as shown in Fig. 2.

[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 when splited into 150 subgraphs.

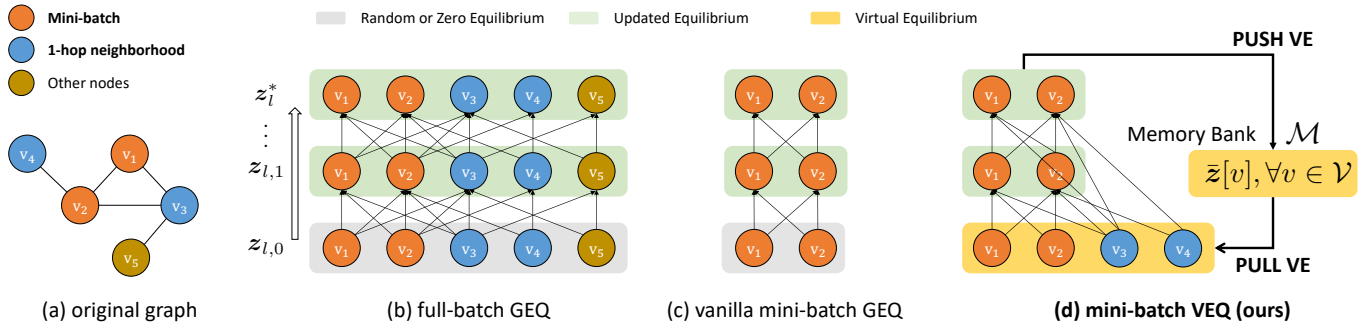

Fig. 3. An illustration of the root-finding computation graph in different training strategies of GEQs. (a) An example graph. (b) Full-batch training, where the final equilibrium $\boldsymbol{z}_l^*$ has global receptive fields. (c) Vanilla mini-batch training on the subgraph of v_1, v_2, which only 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 to update v_1, v_2. We maintain the latest equilibrium of each node v, $\bar{\boldsymbol{z}}[v]$, in a memory bank $\mathcal{M}$. As a result, the updated equilibrium of v_1, v_2 has global receptive fields in VEQ while requiring much less computation compared with full-batch training.

C. 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 [18]–[20] generally assume the contraction of the implicit mapping. As Lemma III.1 shows, the basic assumption guarantees the well-posedness of the fixed point, and enables the application of implicit differentiation in the backward pass. We denote $\nabla_{\boldsymbol{z}}\Phi$ by the gradient of Φ *w.r.t.* $\boldsymbol{z}$, and $\nabla_{\boldsymbol{\theta}}\Phi$ the gradient of Φ *w.r.t.* $\boldsymbol{\theta}$.

Assumption III.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 III.1. *Under Assumption III.1, $\boldsymbol{z}_{\boldsymbol{\theta}}$ is the unique fixed point of $\Phi(\cdot, \boldsymbol{\theta})$, and $\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})$ is invertible.*

Proof. Since $\Phi(\cdot, \boldsymbol{\theta})$ is contraction with constant $q_{\boldsymbol{\theta}}$, based on the Banach fixed-point theorem, $\Phi(\cdot, \boldsymbol{\theta})$ admits a unique fixed point $\boldsymbol{z}_{\boldsymbol{\theta}}$. Moreover, we have

$$\|\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})\| \leq q_{\boldsymbol{\theta}} < 1. \tag{9}$$

Therefore,

$$\left\|(\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta}))^{-1}\right\| = \sum_{k=0}^{\infty} \|\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})\|^k \tag{10}$$

$$\leq \sum_{k=0}^{\infty} q_{\boldsymbol{\theta}}^k = \frac{1}{1 - q_{\boldsymbol{\theta}}}. \tag{11}$$

Thus, $\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})$ 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 III.2. *$\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})$ and $\nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})$ are Lipschitz continuous for every $\boldsymbol{\theta} \in \Lambda$ with constants $L_{\Phi,\boldsymbol{z}}$ and $L_{\Phi,\boldsymbol{\theta}}$, respectively. Moreover, $\|\nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})\|$ is bounded by a constant $C_{\Phi,\boldsymbol{\theta}} > 0$.*

Theorem III.2. *Under Assumptions III.1 and III.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.*

Proof. Based on Eq. (7), we have

$$\nabla_{\boldsymbol{\theta}}\boldsymbol{z} = \nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})(\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta}))^{-1}. \tag{12}$$

From Lemma III.1, we have

$$\|\nabla_{\boldsymbol{\theta}}\boldsymbol{z}\| \leq \left\|(\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}, \boldsymbol{\theta}))^{-1}\right\| \|\nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}, \boldsymbol{\theta})\| \tag{13}$$

$$\leq \frac{C_{\Phi,\boldsymbol{\theta}}}{1 - q} =: C_1. \tag{14}$$

The last inequality uses Assumption III.2. Based on the definition of Lipschitz continuity, we complete the proof. □

Theorem III.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 virtual equilibrium is close to the exact value, the output of VEQ using only 1-hop virtual equilibrium 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 the virtual equilibrium that belongs to $\mathcal{N}[\mathcal{B}]\backslash\mathcal{B}$, and denote $\tilde{\boldsymbol{z}}^{in}$ 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 III.3. *Under Assumptions III.1 and III.2, we have $\left\|\tilde{\boldsymbol{z}}^{in} - \boldsymbol{z}_{\boldsymbol{\theta}}^{in}\right\| \leq C_2 \left\|\boldsymbol{z}^{out} - \boldsymbol{z}_{\boldsymbol{\theta}}^{out}\right\|$, where C_2 is a constant.*

Proof. The point of the proof is that in-batch equilibrium states only depends on its 1-hop neighbors in the update. For a mini-batch $\mathcal{B} \subseteq \mathcal{V}$, let $\tilde{\Phi}(\cdot, \boldsymbol{z}^{out}, \boldsymbol{\theta})$ be the function confined by the out-of-batch node equilibrium $\boldsymbol{z}^{out}$, where $\boldsymbol{z}^{out}$ is taken as its parameter. According to the definition of Φ, we have

$$\boldsymbol{z}_{\boldsymbol{\theta}}^{in} = \tilde{\Phi}(\boldsymbol{z}_{\boldsymbol{\theta}}^{in}, \boldsymbol{z}_{\boldsymbol{\theta}}^{out}, \boldsymbol{\theta}). \tag{15}$$

Also, we have $\|\nabla_{\boldsymbol{z}}\tilde{\Phi}\| < q$, and $\|\nabla_{\boldsymbol{\theta}}\tilde{\Phi}\| < q$. Using the same way we prove Theorem III.2, we have

$$\left\| \frac{\mathrm{d}\boldsymbol{z}^{in}}{\mathrm{d}\boldsymbol{z}^{out}} \right\| \leq \frac{q}{1-q} =: C_2. \tag{16}$$

We complete the proof using the definition of Lipschitz continuity. □

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 virtual equilibrium 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 III.4. *Under Assumptions III.1 and III.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.

Proof. In the following, we denote $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K$ by $\mathbf{A}_K$, and denote $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_{\boldsymbol{\theta}}$ by $\mathbf{A}^*$. To show that $\mathbf{A}_K \to \mathbf{A}^*$, we use the recursive relation of $\mathbf{A}_k$ and $\mathbf{A}^*$ as follows:

$$\begin{aligned}\mathbf{A}_{k+1} =&\mathbf{A}_k(\eta\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) + (1-\eta)\mathbf{I}) + \eta\nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) &(17)\\ =&(1-\eta)\mathbf{A}_k + \eta(\mathbf{A}_k\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta})), &(18)\end{aligned}$$

and

$$\mathbf{A}^* = \mathbf{A}^*\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta}). \tag{19}$$

Based on the recursive relation above and some linear algebra derivation, we get

$$\begin{aligned}&\|\mathbf{A}_{k+1} - \mathbf{A}^*\| \\ \leq &\|(1-\eta)(\mathbf{A}_k - \mathbf{A}^*)\| + \eta\|\nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta})\| \\ &+ \eta\|\mathbf{A}_k\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) - \mathbf{A}^*\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta})\|.\end{aligned} \tag{20}$$

Since

$$\begin{aligned}&\|\mathbf{A}_k\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) - \mathbf{A}^*\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta})\| &(21)\\ \leq &\|\mathbf{A}_k - \mathbf{A}^*\| \cdot \|\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta})\| \\ &+ \|\mathbf{A}^*\| \cdot \|\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta})\| &(22)\\ \leq &q_{\boldsymbol{\theta}}\|\mathbf{A}_k - \mathbf{A}^*\| + C_1\|\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta})\|, &(23)\end{aligned}$$

we have

$$\begin{aligned}&\|\mathbf{A}_{k+1} - \mathbf{A}^*\| &(24)\\ \leq &(1-\eta+\eta q_{\boldsymbol{\theta}})\|\mathbf{A}_k - \mathbf{A}^*\| \\ &+ \eta\|\nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta})\| &(25)\\ &+ \eta C_1\|\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_k,\boldsymbol{\theta}) - \nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}_{\boldsymbol{\theta}},\boldsymbol{\theta})\| \\ \leq &(1-\eta+\eta q_{\boldsymbol{\theta}})\|\mathbf{A}_k - \mathbf{A}^*\| + C_3\eta\|\boldsymbol{z}_k - \boldsymbol{z}_{\boldsymbol{\theta}}\|, &(26)\end{aligned}$$

where $C_3 = L_{\Phi,\boldsymbol{\theta}} + C_1 L_{\Phi,\boldsymbol{z}}$. By iteratively using the relationship, we have

$$\begin{aligned}\|\mathbf{A}_K - \mathbf{A}^*\| \leq &(1-\eta+\eta q_{\boldsymbol{\theta}})^K\|\mathbf{A}_0 - \mathbf{A}^*\| \\ &+ K(1-\eta+\eta q_{\boldsymbol{\theta}})^{K-1} C_3\eta\|\boldsymbol{z}_0 - \boldsymbol{z}_{\boldsymbol{\theta}}\|.\end{aligned} \tag{27}$$

Then it is straight forward to verify that $\mathbf{A}_K \to \mathbf{A}^*$ when $K \to \infty$. □

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.

Assumption III.3. *$\nabla_{\boldsymbol{z}}\ell$ is Lipschitz continuous for every $\boldsymbol{\theta} \in \Lambda$ with a constant $L_{\ell,\boldsymbol{z}}$, and is bounded by a constant $C_{\ell,\boldsymbol{z}} > 0$.*

Theorem III.5. *Suppose Assumptions III.1, III.2 and III.3 hold, and let σ_{max} and σ_{min} be the maximal and minimal singular value of $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}$. If $(\sigma_{min}^2 - \sigma_{max}\|\mathbf{E}\|)\|\boldsymbol{u}\|^2 \geq L_{\ell,\boldsymbol{z}}C_{\ell,\boldsymbol{z}}C_1\|\mathbf{A}_k\|\|\boldsymbol{z}_{\boldsymbol{\theta}} - \boldsymbol{z}_K\|$, then $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}_K$ provides a descent direction of the loss function ℓ.*

Proof. Denote $\mathbf{J} = \nabla_{\boldsymbol{\theta}}\Phi$, $\boldsymbol{v} = \nabla_{\boldsymbol{z}}\ell$, and $\boldsymbol{u} = (\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi)^{-1}\boldsymbol{v}$, $\mathbf{E} = \mathbf{A}_k(\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi) - \nabla_{\boldsymbol{\theta}}\Phi$. Denote the gradient estimate as $\frac{\partial\hat{\ell}}{\partial\boldsymbol{\theta}}$, and the exact gradient as $\frac{\partial\ell}{\partial\boldsymbol{\theta}}$, then

$$\begin{aligned}\left\langle \frac{\partial\hat{\ell}}{\partial\boldsymbol{\theta}}, \frac{\partial\ell}{\partial\boldsymbol{\theta}} \right\rangle =&\left\langle \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\frac{\partial\boldsymbol{z}_K}{\partial\boldsymbol{\theta}}, \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\frac{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}{\partial\boldsymbol{\theta}} \right\rangle \\ &+ \left\langle \left(\frac{\partial\ell}{\partial\boldsymbol{z}_K} - \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\right)\frac{\partial\boldsymbol{z}_K}{\partial\boldsymbol{\theta}}, \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\frac{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}{\partial\boldsymbol{\theta}} \right\rangle,\end{aligned} \tag{28}$$

where

$$\begin{aligned}&\left\langle \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\frac{\partial\boldsymbol{z}_K}{\partial\boldsymbol{\theta}}, \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\frac{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}{\partial\boldsymbol{\theta}} \right\rangle &(29)\\ =&\boldsymbol{v}^\top\mathbf{A}_k^\top\mathbf{J}(\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi)^{-1}\boldsymbol{v} &(30)\\ =&\boldsymbol{u}^\top(\mathbf{I} - \nabla_{\boldsymbol{z}}\Phi)^\top\mathbf{A}_k\mathbf{J}\boldsymbol{u} = \boldsymbol{u}^\top(\mathbf{J} + \mathbf{E})^\top\mathbf{J}\boldsymbol{u} &(31)\\ \geq &\|\mathbf{J}\boldsymbol{u}\|^2 - \|\mathbf{E}\|\|\mathbf{J}\|\|\boldsymbol{u}\| &(32)\\ \geq &(\sigma_{\min}^2 - \sigma_{\max}\|\mathbf{E}\|)\|\boldsymbol{u}\|^2 \geq 0, &(33)\end{aligned}$$

and

$$\begin{aligned}&\left\langle \left(\frac{\partial\ell}{\partial\boldsymbol{z}_K} - \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\right)\frac{\partial\boldsymbol{z}_K}{\partial\boldsymbol{\theta}}, \frac{\partial\ell}{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}\frac{\partial\boldsymbol{z}_{\boldsymbol{\theta}}}{\partial\boldsymbol{\theta}} \right\rangle &(34)\\ \geq &-L_{\ell,\boldsymbol{z}}C_{\ell,\boldsymbol{z}}C_1\|\mathbf{A}_k\|\|\boldsymbol{z}_{\boldsymbol{\theta}} - \boldsymbol{z}_K\|.\end{aligned}$$

We complete the proof by combining them together. □

D. Extension to Multiple Layers

In this section, we first elaborate on the equivalence of multiple IGNN layers and one single wider layer, and then we describe how we implement our VEQ with multiple IGNN layers using the equivalence.

Suppose our implicit layer is composed of T IGNN layers $\Phi_1, \cdots, \Phi_T$ with damping factor η, we use the following theorem to declare that it is actually equivalent to a wider single GEQ layer.

Theorem III.6. *Let $\boldsymbol{z}_0^*$ be the equilibrium of the equation $\boldsymbol{z} = \Phi_T \circ \Phi_{T-1} \cdots \Phi_1(\boldsymbol{z},\boldsymbol{\theta},\mathbf{x})$, let $\boldsymbol{z}_1^* = \Phi_1(\boldsymbol{z}_0^*,\boldsymbol{\theta},\mathbf{x})$, $\boldsymbol{z}_2^* =$*

$\Phi_2 \circ \Phi_1(\boldsymbol{z}_0^*, \boldsymbol{\theta}, \mathbf{x})$, $\cdots$, $\boldsymbol{z}_{T-1}^* = \Phi_{T-1} \circ \cdots \circ \Phi_2 \circ \Phi_1(\boldsymbol{z}_0^*, \boldsymbol{\theta}, \mathbf{x})$, *then* $\tilde{\boldsymbol{z}}^* = (\boldsymbol{z}_1^* \oplus \cdots \oplus \boldsymbol{z}_{T-1}^* \oplus \boldsymbol{z}_0^*)^\top$ *is the equilibrium of the equation:*

$$\tilde{\boldsymbol{z}} = \eta\sigma(\mathbf{P}^\top \tilde{\mathbf{W}}^\top \otimes \hat{\mathbf{A}}\tilde{\boldsymbol{z}} + \tilde{\mathbf{U}} \otimes \mathbf{I}\boldsymbol{x}) + (1-\eta)\mathbf{P}\tilde{\boldsymbol{z}}, \tag{35}$$

where $\tilde{\mathbf{W}}$ *is block diagonal,* $\mathbf{P}$ *is a permutation matrix, and* $\tilde{\mathbf{U}}$ *is a concatenated matrix as follows:*

$$\tilde{\mathbf{W}} := \begin{bmatrix} \mathbf{W}_1 & & & \\ & \mathbf{W}_2 & & \\ & & \ddots & \\ & & & \mathbf{W}_T \end{bmatrix}, \tag{36}$$

$$\mathbf{P} := \begin{bmatrix} \mathbf{0} & & & & \mathbf{I} \\ \mathbf{I} & \mathbf{0} & & & \\ & \ddots & \ddots & & \\ & & \ddots & \mathbf{0} & \\ & & & \mathbf{I} & \mathbf{0} \end{bmatrix}, \ \tilde{\mathbf{U}} = \begin{bmatrix} \mathbf{U}_1^\top \\ \mathbf{U}_2^\top \\ \vdots \\ \mathbf{U}_{T-1}^\top \\ \mathbf{U}_T^\top \end{bmatrix}. \tag{37}$$

Proof. We can rewrite $\tilde{\boldsymbol{z}} = \Phi_T \circ \Phi_{T-1} \cdots \Phi_1(\tilde{\boldsymbol{z}}, \boldsymbol{\theta}, \mathbf{x})$ in a separated matrix form:

$$\begin{aligned} \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \vdots \\ \boldsymbol{z}_{T-1} \\ \boldsymbol{z}_0 \end{bmatrix} =& \eta\sigma\left(\begin{bmatrix} \mathbf{0} & & & & \mathbf{W}_1^\top \\ \mathbf{W}_2^\top & \mathbf{0} & & & \\ & \ddots & \ddots & & \\ & & \ddots & \mathbf{0} & \\ & & & \mathbf{W}_T^\top & \mathbf{0} \end{bmatrix} \right. \\ & \left. \otimes \hat{\mathbf{A}} \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \vdots \\ \boldsymbol{z}_{T-1} \\ \boldsymbol{z}_0 \end{bmatrix} + \begin{bmatrix} \mathbf{U}_1^\top \\ \mathbf{U}_2^\top \\ \vdots \\ \mathbf{U}_{T-1}^\top \\ \mathbf{U}_T^\top \end{bmatrix} \otimes \mathbf{I}\boldsymbol{x} \right) \\ & + (1-\eta) \begin{bmatrix} \mathbf{0} & & & & \mathbf{I} \\ \mathbf{I} & \mathbf{0} & & & \\ & \ddots & \ddots & & \\ & & \ddots & \mathbf{0} & \\ & & & \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \vdots \\ \boldsymbol{z}_{T-1} \\ \boldsymbol{z}_0 \end{bmatrix}. \end{aligned} \tag{38}$$

Hence a multi-layer VEQ is actually a single-layer VEQ with multiple layers. □

Implementation of Multi-layer VEQ With the above theorem, we notice that an evaluation of the multi-layer function $\Phi_T \circ \Phi_{T-1} \cdots \Phi_1(\cdot, \boldsymbol{\theta}, \mathbf{x})$ applied on a node v still only accesses its 1-hop neighbors. As a result, our discussions on the 1-layer case also apply to the multi-layer case. For a mini-batch $\mathcal{B}$, we use the superscript "in" and "out" to denote the node equilibrium that belongs to $\mathcal{B}$ and $\mathcal{N}[\mathcal{B}]\backslash\mathcal{B}$, respectively. In the implementation, we notice that the in-batch equilibrium $\boldsymbol{z}_1^{in}, \cdots, \boldsymbol{z}_{T-1}^{in}$ can also be derived sequentially from $\boldsymbol{z}_0^{in}$ and $\tilde{\boldsymbol{z}}^{out}$. Consequently, to reduce data transfer, we pull $\boldsymbol{z}_0^*[\mathcal{B}]$ from the memory bank for in-batch nodes, and $\tilde{\boldsymbol{z}}^*[\mathcal{N}[\mathcal{B}]\backslash\mathcal{B}]$ for out-of-batch nodes. And when we finish fixed-point iterations, we push back the in-batch equilibrium $\tilde{\boldsymbol{z}}^*[\mathcal{B}]$, which serves as virtual equilibrium in the next model update. Between fixed-point iterations, we do not need to push or pull any virtual equilibrium.

IV. Related Work

A. Graph Equilibrium Models

DEQs (Deep Equilibrium Models) [25], [30] are a new kind of implicit models, whose output is the solution to a fixed-point equation. IGNN [18] 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 [20] and EIGNN [19], 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 decomposition 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 [31] and Phantom Gradient [29] 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, so that it would take many more steps to approximate the exact gradient. The Phantom Gradient starts unrolling from the equilibrium. Although it can approximate $\nabla_{\boldsymbol{\theta}}\boldsymbol{z}$ 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-finding process before the unrolling starts.

B. Sampling-based Graph Neural Networks

Explicit GNNs suffer from the neighbor explosion phenomenon, where the required nodes increase exponentially over layers [22]. Various sampling-based methods have been explored to alleviate this issue, including node-wise sampling approaches [6], [22], [32], layer-wise sampling approaches [23], [33], [34] and subgraph-based sampling approaches [24], [35], [36]. Several recent works also study reusing the historical node embeddings from previous training steps to approximate full-batch training, such as GAS [37] and GraphFM [38]. As far as we know, we are the first to train GEQs in mini-batch on large-scale graphs with full-batch information. Compared with these works, 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 push and pull historical embeddings each time they update a layer ($\mathcal{O}(L)$). Instead, in our VEQ, we only need to push and pull the equilibrium $\boldsymbol{z}$

TABLE I
STATISTICS AND PROPERTIES OF THE DATASETS. THE "M" DENOTES THE MULTI-LABEL CLASSIFICATION TASK, AND "S" DENOTES THE SINGLE LABEL CLASSIFICATION TASK.

Dataset	Nodes	Edges	Degree	Classes	Train/Validation/Test
Flickr	89, 250	899, 756	10	7(s)	0.50/0.25/0.25
Reddit	232, 965	114, 615, 892	50	41(s)	0.66/0.10/0.24
Yelp	716, 847	13, 954, 819	10	100(m)	0.75/0.10/0.15
ogbn-arxiv	169,343	1,166,243	9	40(s)	0.537/0.176/0.287
ogbn-products	2,449,029	61,859,140	25	47(s)	0.10/0.02/0.88
PPI	56,994	818, 716	14	121(m)	0.79/0.11/0.10

TABLE II
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	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%	<u>99.36</u>
	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	<u>54.46</u>	95.40	-	-
Implicit	IGNN	53.40±0.1%*	94.54±0.1%*	63.58±0.1%*	97.60
	VEQ	**55.34±0.7%**	<u>96.81±0.3%</u>	<u>64.90±0.1%</u>	99.22±0.2%

TABLE III
NODE CLASSIFICATION RESULTS ON LARGE-SCALE OGB DATASETS: MICRO-F1 SCORE (%). WE MARK THE RESULTS IMPLEMENTED BY US WITH $*$. OOM: OUT OF MEMORY.

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	-	<u>79.08</u>
	GAS	71.68	76.66
	GraphFM	<u>71.81</u>	76.88
Implicit	IGNN	70.98*	OOM*
	VEQ	**72.82**	**79.23**

once in the memory bank ($\mathcal{O}(1)$), which largely reduces the overhead caused by data transfer.

V. EXPERIMENTS

In this section, we conduct a comprehensive analysis of VEQ on large-scale benchmark datasets. Our code is available at https://github.com/7qchen/VEQ.

A. Performance on Benchmark Datasets

Hardware and Softwares We conduct our experiments on NVIDIA GeForce RTX 3090 Ti with 24 GB memory, and Intel Xeon Gold 6342 CPU. And we implement our VEQ based on PyTorch [39], Pytorch_Geometric [40], and PyGAS [37].

Datasets We study six large-scale graphs with thousands or millions of nodes and edges, including Flickr [35], Yelp [35], Reddit [22], PPI [22], and two from OGB benchmark [41]: ogbn-arxiv and ogbn-products. The statistics for the six benchmark graphs is listed in Table I. Among them, PPI is the only dataset that contains multiple graphs, and Yelp is the only dataset where a node is associated with multiple labels.

Baselines We compare VEQ against several representative explicit and implicit models. For explicit models, we consider two node-wise sampling methods FastGCN [23] and GraphSAGE [22], a layer-wise sampling method VR-GCN [32], and two subgraph-sampling methods Cluster-GCN [24] and GraphSAINT [35]. Besides, we also include two historical-based scalable methods: GAS [37] and GraphFM [38]. We report their GCN-based [4] variants for a fair comparison, which share the same aggregation mechanism as ours. For implicit models, we adopt IGNN [18] as our baseline. EIGNN [19] is not included, as it barely works on large-scale datasets. For example, on the smallest Flickr dataset, its preprocessing step cannot even finish in 8 hours ($10\times$ IGNN's total training time). We do not include CGS [20] as it is only designed for graph classification tasks. Since the authors of IGNN [18] 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 II. We also report the results of two OGB datasets in Table III. 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,

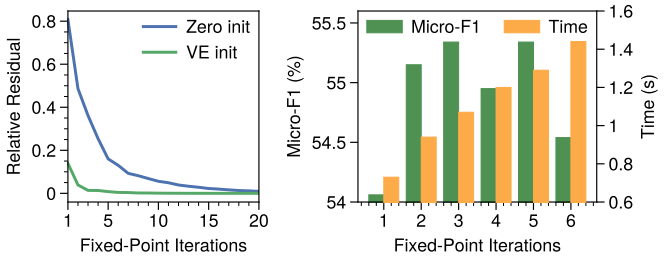

Fig. 4. Left: the relative residual of root-finding with different fixed-point iterations on Flickr using zero or virtual equilibrium as initialization. Right: micro-F1 score (%) and per-epoch training time with different fixed-point iterations on Flickr.

TABLE IV
ABLATION STUDY OF DIFFERENT USAGES OF VIRTUAL EQUILIBRIUM (VE) ON FLICKR: MICRO-F1 SCORE (%).

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**

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 with full-batch training simulators like GAS and GraphFM. This is mainly because our VEQ enjoys global receptive fields with persistent root-finding along training, while an L-layer explicit model can only capture information within L-hop neighbors.

B. Equilibrium Computation

To verify that VE initialization indeed helps to find the equilibrium more efficiently, we show the relative residual $\|\boldsymbol{z}_{i+1} - \boldsymbol{z}_i\| / \|\boldsymbol{z}_i\|$ along the root-finding process in Fig. 4 (left). We can see that our model exhibits stable convergence with both zero and VE initialization, while the latter converges significantly faster than the former. Indeed, as shown in Fig. 4 (right), a small iteration number K is enough to achieve 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. Too few iterations may not be enough for equilibrium updates, while too many iterations may intensify the change of equilibrium in case the model does not converge well. Generally, $K = 3$ is an appropriate choice for Flickr, and it is still much faster than IGNN (*c.f.* Fig. 1).

TABLE V
TOTAL GPU MEMORY COST (IN GB) AND THE PROPORTION OF DATA USED FOR COMPUTING THE EQUILIBRIUM ON FLICKR.

#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%

C. Ablation Study

We further conduct an ablation study on the two roles of VE: 1) initialization for in-batch nodes, and 2) reference points for out-of-batch nodes, under both full-batch and mini-batch settings. From Table IV, 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 V shows that VEQ also has less memory consumption under mini-batch training while it utilizes 100% information with global receptive fields.

VI. 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-finding 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

[1] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.
[2] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.
[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.
[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Machine Learning*, 2016.

[5] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[6] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[7] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, 2019.

[8] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *ESWC*. Springer, 2018, pp. 593–607.

[9] N. Park, A. Kan, X. L. Dong, T. Zhao, and C. Faloutsos, "Estimating node importance in knowledge graphs using graph neural networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 596–606.

[10] H. Ren, W. Hu, and J. Leskovec, "Query2box: Reasoning over knowledge graphs in vector space using box embeddings," in *International Conference on Machine Learning*, 2019.

[11] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*, 2017.

[13] Y.-C. Lo, S. E. Rensi, W. Torng, and R. B. Altman, "Machine learning in chemoinformatics and drug discovery," *Drug Discovery Today*, vol. 23, no. 8, pp. 1538–1546, 2018.

[14] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann *et al.*, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, 2020.

[15] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[16] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *International Conference on Machine Learning*, 2019.

[17] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "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, 2020, pp. 3438–3445.

[18] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui, "Implicit graph neural networks," in *Advances in Neural Information Processing Systems*, 2020.

[19] J. Liu, K. Kawaguchi, B. Hooi, Y. Wang, and X. Xiao, "EIGNN: Efficient infinite-depth graph neural networks," in *Advances in Neural Information Processing Systems*, 2021.

[20] J. Park, J. Choo, and J. Park, "Convergent graph solvers," in *International Conference on Machine Learning*, 2021.

[21] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of Computation*, vol. 19, no. 92, pp. 577–593, 1965.

[22] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017.

[23] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Machine Learning*, 2018.

[24] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

[25] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," in *Advances in Neural Information Processing Systems*, 2019.

[26] X. Xie, Q. Wang, Z. Ling, X. Li, G. Liu, and Z. Lin, "Optimization induced equilibrium networks: An explicit optimization perspective for understanding equilibrium models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[27] D. G. Anderson, "Iterative procedures for nonlinear integral equations," *Journal of the ACM*, vol. 12, no. 4, pp. 547–560, 1965.

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

[29] Z. Geng, X.-Y. Zhang, S. Bai, Y. Wang, and Z. Lin, "On training implicit models," *Advances in Neural Information Processing Systems*, 2021.

[30] S. Bai, V. Koltun, and J. Z. Kolter, "Multiscale deep equilibrium models," in *Advances in Neural Information Processing Systems*, 2020.

[31] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo, "On the iteration complexity of hypergradient computation," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3748–3758.

[32] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *International Conference on Machine Learning*. PMLR, 2018, pp. 942–950.

[33] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[34] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[35] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graph-SAINT: Graph sampling based inductive learning method," in *International Conference on Machine Learning*, 2019.

[36] W. Cong, R. Forsati, M. Kandemir, and M. Mahdavi, "Minimal variance sampling with provable guarantees for fast training of graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1393–1403.

[37] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec, "GNNAutoScale: Scalable and expressive graph neural networks via historical embeddings," in *International Conference on Machine Learning*. PMLR, 2021, pp. 3294–3304.

[38] H. Yu, L. Wang, B. Wang, M. Liu, T. Yang, and S. Ji, "GraphFM: Improving large-scale GNN training via feature momentum," in *International Conference on Machine Learning*. PMLR, 2022, pp. 25 684–25 701.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[40] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *International Conference on Machine Learning Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[41] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "OGB-LSC: A large-scale challenge for machine learning on graphs," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.