IGNN-Solver: A Graph Neural Solver for Implicit Graph Neural Networks

Junchao Lin, Zenan Ling, *Member, IEEE*, Zhanbo Feng, *Student Member, IEEE*, Jingwen Xu, Minxuan Liao, Feng Zhou, Tianqi Hou, Zhenyu Liao, *Member, IEEE*, and Robert C. Qiu, *Fellow, IEEE*

***Abstract*—Implicit graph neural networks (IGNNs), which exhibit strong expressive power with a single layer, have recently demonstrated remarkable performance in capturing long-range dependencies (LRD) in underlying graphs while effectively mitigating the over-smoothing problem. However, IGNNs rely on computationally expensive fixed-point iterations, which lead to significant speed and scalability limitations, hindering their application to large-scale graphs. To achieve fast fixed-point solving for IGNNs, we propose a novel graph neural solver, IGNN-Solver, which leverages the generalized Anderson Acceleration method, parameterized by a tiny GNN, and learns iterative updates as a graph-dependent temporal process. To improve effectiveness on large-scale graph tasks, we further integrate sparsification and storage compression methods, specifically tailored for the IGNN-Solver, into its design. Distinct from previous solvers and IGNN variants, IGNN-Solver treats the solver itself as a learnable, graph-aware dynamical system, enabling solver-level acceleration. Extensive experiments demonstrate that the IGNN-Solver significantly accelerates inference on both small- and large-scale tasks, achieving a $1.5\times$ to $8\times$ speedup without sacrificing accuracy. This advantage becomes more pronounced as the graph scale grows, facilitating its large-scale deployment in real-world applications. The code to reproduce our results is available at https://github.com/landrarwolf/IGNN-Solver.**

***Index Terms*—Implicit graph neural networks, large-scale graphs, fixed-point solver, deep equilibrium models.**

I. Introduction

IMPLICIT graph neural networks (IGNNs) [1–3] have emerged as a significant advancement in graph learning frameworks. Unlike traditional graph neural networks (GNNs) that stack multiple explicit layers, IGNNs utilize a single implicit layer formulated as a fixed-point equation. The solution to this fixed-point equation, known as the equilibrium, is equivalent to the output obtained by iterating an explicit layer infinitely. This allows the implicit layer to access infinite hops of neighbors, providing IGNNs with global receptive fields within just one layer [4]. As a result, IGNNs effectively address the long-standing issue of over-smoothing in conventional explicit GNNs and capture long-range dependencies in graph-structured data [5–7].

However, existing IGNNs suffer from slow speeds and have difficulty scaling to large-scale graphs [8–11]. This is primarily because IGNNs derive features by solving for fixed points, demanding substantial computational resources. For example, even on the Citeseer dataset [12] classification task with a small-scale graph, IGNNs require more than 20 forward iterative computations to nearly converge to a fixed point [1]. The computational overhead of solving fixed points is amplified by the task scale, resulting in notably slow inference speeds compared to explicit GNNs. This substantial drawback poses challenges for IGNNs in generalizing to large-scale graphs in practical scenarios.

In response to this challenge, we propose a novel graph neural solver for IGNNs, termed IGNN-Solver. It takes the IGNN layer as input, which includes the graph information matrix and layer parameters, and outputs the solution to the corresponding fixed-point equation. In contrast to existing IGNN variants [2, 3, 13, 14] and structure-agnostic solvers [15], IGNN-Solver learns to reshape the trajectory via a graph-aware strategy. Consequently, the proposed IGNN-Solver remarkably accelerates the model's inference speed without compromising accuracy and with only a slight increase in training overhead. This advantage becomes increasingly pronounced as the scale of the graph grows, making it particularly beneficial for deploying IGNNs in large-scale graph tasks.

Our IGNN-Solver comprises two components. First, we introduce a learnable initializer that estimates an optimal initial point for the optimization process. Second, we propose a generalized version of Anderson Acceleration (AA) [16], employing a tiny graph neural network to model the iterative updates as a sequence of graph-dependent steps. Compared to the solvers proposed for conventional Implicit Neural Networks (INNs) [15, 17, 18], we introduce a novel improvement by learning solver parameters through a graph-structured formulation with well-posedness guarantees. This approach circumvents the potential loss of graph information, thereby improving the model's performance. Moreover, IGNN-Solver has significantly fewer parameters compared to IGNN, and its training is independent of the IGNN inference process. Consequently, the training of IGNN-Solver proceeds rapidly without sacrificing generalization.

Manuscript created on February 17, 2025. The work of Zenan Ling is supported in part by the National Natural Science Foundation of China (via NSFC-62406119), the Natural Science Foundation of Hubei Province (2024AFB074), and the Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010001). The work of Zhenyu Liao is supported in part by the National Natural Science Foundation of China (via NSFC-62206101). The work of Robert Caiming Qiu is supported in part by the National Natural Science Foundation of China (via NSFC-12141107) and in part by the Key Research and Development Program of Guangxi (GuiKe-AB21196034). *(Corresponding author: Zenan Ling.)*

Junchao Lin, Zenan Ling, Minxuan Liao, Zhenyu Liao, and Robert C. Qiu are with the School of EIC, Huazhong University of Science and Technology.

Jingwen Xu is with the School of Science, Wuhan University of Technology.

Zhanbo Feng is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University.

Feng Zhou is with the Center for Applied Statistics and School of Statistics, Renmin University of China.

Tianqi Hou is with the Lagrange Mathematics and Computing Research Center of Huawei, Paris.

Moreover, to address the challenges posed by large-scale graphs, we incorporate both graph sparsification and storage compression in the solver design. Specifically, to prevent the network from becoming excessively large and deviating from the initial goal of using a compact network for prediction [19, 20], we utilize graph sparsification to obtain a lighter graph. Additionally, directly mapping data from high to extremely low dimensions is not appropriate [21]; therefore, we employ a storage compression technique to mitigate this issue.

In our experiments, we apply IGNN-Solver to extensive real-world datasets from diverse domains and scales, including five large-scale datasets: Amazon-all [22], Reddit [23], ogbn-arxiv [24], ogbn-products [24], and ogbn-papers100M [24]. Our results demonstrate that IGNN-Solver achieves higher accuracy with reduced inference time, showing a $1.5\times$–$8\times$ speedup, and incurs minimal additional training overhead, constituting only 1% of the overall training time, while requiring a comparable level of GPU memory.

Our main contributions are summarized as follows:

- We introduce IGNN-Solver, a learnable method designed to predict the next fixed-point iteration of the IGNN layer through a well-posed iterative formulation. This innovative approach mitigates the need for extensive iterations, common in vanilla IGNNs, thereby substantially accelerating inference while preserving accuracy and minimizing parameter consumption.
- We formulate the solver itself as a *graph-aware* learnable module. Instead of relying on structure-agnostic acceleration modules or predefined, non-learnable solver trajectories, IGNN-Solver adapts the update rule to node connectivity and local graph context. This enables *graph-conditioned reshaping* of the fixed-point iteration trajectory, providing a principled and efficient solver design.
- We design tailored sparsification and storage-compression techniques and integrate them into the graph-aware solver architecture. These techniques address the computational and memory challenges that arise when scaling learnable implicit solvers to large graphs, while remaining compatible with the original IGNN formulation. As a result, the proposed solver remains efficient and memory-friendly even on large-scale graphs, enabling practical deployment without substantial additional cost.
- We validate our approach through comprehensive experiments on over 20 real-world datasets, covering four types of graph data: standard graphs (including small [12, 25–28], large [22, 24, 29], and ultra-scale [30]), heterogeneous graph [31], dynamic graph [32], and hypergraph [33]. These datasets further span multiple learning scenarios, including node classification, graph classification, multi-label classification, and dynamic graph classification. Our results demonstrate that IGNN-Solver incurs minimal computational overhead (approximately 1% of the total) and achieves up to an 8-fold increase in inference speed without compromising accuracy.

II. Related Works

The typical GNN [12] and its variants [23, 34] have been widely used for graph data modeling in various tasks. Different GNNs have been proposed to utilize attention mechanism [34], neighbors sampling [23], pseudo-coordinates [35] and graph fusion [36]. However, due to issues such as over-smoothing, depth, and bottlenecks, these models typically involve finite aggregation layers [5–7]. To address it, recent works [1–3] have developed Implicit Graph Neural Networks, encouraging these models to capture long-range dependencies on graphs. Here, we highlight the contributions of our proposed IGNN-Solver through a detailed comparison with Implicit Graph Neural Networks (IGNNs) and Deep Equilibrium Models (DEQs), both of which are closely related to our approach.

A. Implicit Graph Neural Networks

Instead of stacking a series of operators hierarchically, implicit GNNs define their outputs as solutions to nonlinear dynamical systems, which is initially introduced by [1] to tackle challenges associated with learning long-range dependencies in graphs. [8] proposes a new implicit graph model enabling mini-batch training without sacrificing the ability to capture long-range dependencies. [3] theoretically investigates the well-posedness of the IGNN model from a monotone operator viewpoint. Subsequently, [13] introduces a novel approach based on multiscale implicit layers to capture graph structures at different resolutions. [14] proposes a mini-batch training strategy and a stochastic solver to incorporate global and long-range information.

Although the aforementioned IGNN works well by alleviating the problem of over-smoothing of features by allowing meaningful fixed points to propagate implicitly, the inherent *slow inference speed* of implicit networks poses a major obstacle to its scalability. The main reason is that the solver of the fixed-point network is inefficient (e.g., the Picard solver used by IGNN [1] and Anderson acceleration solver used by MIGNN [3]), makes the overhead of the fixed-point solver magnified by the task scales. In comparison, by enabling the acceleration strategy itself to be *graph-aware*, our proposed IGNN-Solver reshapes the iteration trajectory to accelerate the iteration solving process of IGNNs, which addresses *the most limiting drawback* compared to traditional feedforward models.

Another class of IGNNs based on Neural ODEs [37] has emerged to address issues like depth and bottlenecks. For example, [38] models continuous residual layers using GCNs. [39] proposes methods for modeling static and dynamic graphs with GCNs, along with a hybrid approach where latent states evolve continuously between RNN steps in dynamic graphs. [40] tackles the problem of continuous message passing. [41] introduces a graph neural diffusion network based on the discretization of diffusion PDEs on graphs. [42] enhances graph neural diffusion with a source term and connects the model to random walk formulation on graphs. However, they essentially view deep learning on graphs as a continuous diffusion process, which differs from IGNNs based on fixed-point networks, thus requiring manually tuning the termination time and step size of the diffusion equation. In contrast, our IGNN-Solver uses implicit formulations instead of explicit diffusion discretization, which admits an equilibrium corresponding to infinite diffusion steps and expands the receptive field.

B. Fixed-point Solvers for Deep Equilibrium Models

Traditional deep learning models use multi-layered networks, while DEQs [17] find a fixed-point of a single layer, representing the network's equilibrium state. Viewed as infinitely deep networks, DEQs use root-finding for forward propagation and implicit differentiation for backward propagation. Therefore, DEQs capture complex patterns with constant memory and computational costs [43]. Monotone operator theory has been used to guarantee the convergence of DEQs [44] and to improve the stability of implicit networks [15].

However, it is well-known that DEQs, as typical implicit models, suffer from slow training and inference speeds [8, 45], which is highly disadvantageous for large-scale scenarios like GraphGPT [46]. To alleviate this issue, recent efforts have explored certain improved solver methods for DEQs, further optimizing root-finding problems and making these models easier to solve. [47] discusses the amortization of the cost of the iterative solver that would otherwise make implicit models slow. [9] proposes a novel gradient approximation method for implicit models, which significantly accelerates the backward passes in training implicit models. [15] discusses the superiority of learnable solvers over generic solvers in implicit models using a tiny neural network and significantly enhances the efficiency of such models through custom neural solvers.

In contrast, IGNN has a similar network representation to DEQ, both defining the output as *the solution of the equation* to obtain network outputs. But the difference lies in the fact that IGNN's equilibrium equations encode graph structure while DEQ does not, which will undoubtedly deepen its weakness of *slow inference speed* and make IGNN's solution slower. Insight on it, our proposed IGNN-Solver leverages graph information to guide solver acceleration, achieving *fast and meaningful* implicit graph network propagation, especially in the case of graph data being large-scale.

III. Preliminaries

A. Explicit GNNs

Let $\mathcal{G} = \{\boldsymbol{A}, \boldsymbol{X}\}$ represent an undirected graph, where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix indicating the relationships between the nodes in $\mathcal{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}$, and n is the number of nodes. The node feature matrix $\boldsymbol{X} = [\boldsymbol{x}_1; \boldsymbol{x}_2; \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d\times n}$ contains the features of the nodes, with d representing the feature dimension. Conventional (explicit) GNNs [12, 34] feature a learnable aggregation process centered on the message-passing operation within the graph [48]. This process iteratively propagates information from each node to its neighboring nodes. The formal general structure for each layer l is defined as follows:

$$\boldsymbol{H}^{[l+1]} = f_\theta(\boldsymbol{A}, \boldsymbol{H}^{[l]}), \quad \boldsymbol{H}^{[0]} = \boldsymbol{X}, \tag{1}$$

where $\boldsymbol{H}^{[l]}$ represents the hidden node representation, f_θ denotes the parameters in l-th layer. A commonly used GNN is Graph Convolutional Network (GCN) [12], defined as $\boldsymbol{H}^{[l+1]} = \sigma(\widehat{\boldsymbol{A}}\boldsymbol{H}^{[l]}\boldsymbol{W}^{[l]})$, where $\boldsymbol{W}^{[l]}$ denotes the weight matrix of the l-th layer, $\sigma(\cdot)$ denotes a non-expansive activation function, and $\widehat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1/2}(\boldsymbol{A}+\boldsymbol{I})\tilde{\boldsymbol{D}}^{-1/2}$ represents the symmetric normalized graph matrix. $\tilde{\boldsymbol{D}}$ is a diagonal matrix with $\tilde{\boldsymbol{D}}_{ii} = 1 + \sum_j \boldsymbol{A}_{ij}$. GNNs leverage the above message-passing operation in (1) to learn useful information. Still, they often involve a limited number of l layers due to over-smoothing [6], making it challenging for GNNs to capture the long-range dependency (LRD) on graphs.

B. Implicit GNNs

1) Architecture: Similar to traditional explicit GNNs, IGNNs [1–4] also involve an aggregation process, with the distinction that the depth of layers (iteration step k) is *infinite*. The aggregation process in IGNNs is typically defined as

$$\boldsymbol{Z}^{[k+1]} = \sigma(\boldsymbol{W}\boldsymbol{Z}^{[k]}\widehat{\boldsymbol{A}} + b_\Omega(\boldsymbol{X})), k = 1, 2, \ldots, \infty, \tag{2}$$

where b_Ω represents affine transformation parameterized by $\boldsymbol{\Omega}$, and the weight matrices $\boldsymbol{W}$ and $\boldsymbol{\Omega}$ are globally shared at each iteration step. $\sigma(\cdot)$ denotes a non-expansive activation function. The IGNN model f_θ is formally described by

$$\boldsymbol{Z}^\star = f_\theta(\boldsymbol{Z}^\star, \widehat{\boldsymbol{A}}, \boldsymbol{X}) = \sigma(\boldsymbol{W}\boldsymbol{Z}^\star\widehat{\boldsymbol{A}} + b_\Omega(\boldsymbol{X})), \tag{3}$$

where the representation, given as the "internal state" $\boldsymbol{Z}^\star$, is obtained as the fixed-point solution of the equilibrium (2). The final representation theoretically encompasses information from all neighbors in the graph. Therefore, IGNNs capture LRD within the graph, offering better performance compared to GNNs with finite iterations [1, 2]. Another notable advantage of this framework is its memory efficiency, as it only needs to retain the current state $\boldsymbol{Z}$ without requiring additional intermediate representations.

2) Well-posedness: Notably, to obtain a valid representation for any given input $\boldsymbol{X}$ from IGNNs, we need to solve the unique internal state $\boldsymbol{Z}^\star$ in (3). However, there may not always exist a well-defined solution $\boldsymbol{Z}^\star$ for some inputs $\boldsymbol{X}$ [1]. To guarantee the existence and uniqueness of the solution to (3), we introduce the concept of *well-posedness* for IGNNs f_θ with activation function σ [1]. Specifically, the weight matrix $\boldsymbol{W} \in \mathbb{R}^{m\times m}$ and the adjacency matrix $\widehat{\boldsymbol{A}} \in \mathbb{R}^{n\times n}$ are said to be well-posed for σ if the solution $\boldsymbol{Z}^\star \in \mathbb{R}^{m\times n}$ of (3) exists and is unique for any $b_\Omega(\boldsymbol{X}) \in \mathbb{R}^{m\times n}$.

Based on the *monotone operator theorem* [44, 49], a sufficient condition for the well-posedness for IGNNs is defined as follows. Let the nonlinearity σ be a non-expansive activation function (*e.g.*, Sigmoid, Tanh, ReLU, *etc.*), and let $\widehat{\boldsymbol{W}} = \widehat{\boldsymbol{A}}^\top \otimes \boldsymbol{W}$. [1] Then, the IGNN model defined in (3) is well-posed as long as $\boldsymbol{I} - \widehat{\boldsymbol{W}} \succeq m\boldsymbol{I}$ for some $m > 0$. [2]

It establishes that if $\widehat{\boldsymbol{W}}$ is symmetric and $\boldsymbol{I} - \widehat{\boldsymbol{W}} \succeq m\boldsymbol{I}$ for some $m > 0$, then the existence and uniqueness of the equilibrium point $\boldsymbol{Z}^\star$ are guaranteed, ensuring the model is well-posed. This follows from the sufficient condition that $\boldsymbol{I} - \widehat{\boldsymbol{W}}$ is strongly monotone.

To satisfy the well-posedness of IGNNs, a commonly used method [3, 50], is to orthogonally reparameterize $\boldsymbol{W}$ in the

[1] We represent the Kronecker product of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as $\boldsymbol{A} \otimes \boldsymbol{B}$.

[2] We represent $\boldsymbol{A} \succeq \boldsymbol{B}$ if $\boldsymbol{A} - \boldsymbol{B}$ is semi-positive definite. For the case of non-symmetric matrices $\widehat{\boldsymbol{W}}$, positive definiteness can be defined as the positive definiteness of the symmetric component, i.e., $\boldsymbol{I} - \widehat{\boldsymbol{W}} \succeq m\boldsymbol{I}$ if and only if $\boldsymbol{I} - \frac{\widehat{\boldsymbol{W}} + \widehat{\boldsymbol{W}}^T}{2} \succeq m\boldsymbol{I}$.

Algorithm 1 Anderson Acceleration Solver

Input: fixed-point function $f_\theta : \mathbb{R}^n \to \mathbb{R}^n$, max storage size m, residual control parameter β

1: Set initial point value $\boldsymbol{Z}^{[0]} \in \mathbb{R}^n$ ▷ set $\mathbf{0}$ or random value normally
2: **for** $k = 0, \ldots, K-1$ **do**
3: 1) Set $m_k = \min\{m, k\}$ ▷ storage of the most recent steps
4: 2) Solve $\boldsymbol{\alpha}^{[k]} = \arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^{m_k+1}} \left\| \boldsymbol{G}^{[k]} \boldsymbol{\alpha} \right\|_F$, s.t. $\mathbf{1}^\top \boldsymbol{\alpha}^{[k]} = 1$ ▷ compute weights
5: 3) $\boldsymbol{Z}^{[k+1]} = \beta \sum_{i=0}^{m_k} \boldsymbol{\alpha}_i^{[k]} f_\theta(\boldsymbol{Z}^{[k-m_k+i]}) + (1-\beta) \sum_{i=0}^{m_k} \boldsymbol{\alpha}_i^{[k]} \boldsymbol{Z}^{[k-m_k+i]}$ ▷ Anderson step
6: **end for**

IGNN layer f_θ using the following Cayley map, which is inspired from the unitary operator [51], as

$$\boldsymbol{W} = \kappa(\boldsymbol{I} - \boldsymbol{S})(\boldsymbol{I} + \boldsymbol{S})^{-1}, \tag{4}$$

where $\kappa \in (0, 1)$ and $\boldsymbol{S} = \boldsymbol{C} - \boldsymbol{C}^\top$ is a skew-symmetric matrix with $\boldsymbol{C}$ as an arbitrary parameterized matrix. From (4), it follows that $\boldsymbol{W}$ is positive definite and its real eigenvalues are in $(0, 1)$. Moreover, based on the definition of $\widehat{\boldsymbol{A}}$, we can derive the specatral norm of $\widehat{\boldsymbol{W}}$ are in $(0, 1)$, which ensures that $\boldsymbol{I} - \widehat{\boldsymbol{W}} \succeq m\boldsymbol{I}$, thereby ensuring the well-posedness of IGNN. We refer readers to Appendix A for detailed proof.

C. Training of IGNNs

Additionally, to train the parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{\Omega}\}$ during the back-propagation process of neural networks, IGNNs compute the Jacobian $\nabla_{\boldsymbol{Z}} \mathcal{L}$ by solving the equilibrium equation, as derived from the implicit differential method and the *implicit function theorem* [52]:

$$\nabla_{\boldsymbol{Z}} \mathcal{L} = \boldsymbol{D} \odot (\boldsymbol{W}^\top \nabla_{\boldsymbol{Z}} \mathcal{L} \widehat{\boldsymbol{A}}^\top + \nabla_{\boldsymbol{X}} \mathcal{L}), \tag{5}$$

where $\boldsymbol{D} = \sigma'(\boldsymbol{W}\boldsymbol{X}\widehat{\boldsymbol{A}} + b_\Omega(\boldsymbol{U}))$ and $\sigma'(\cdot)$ refers to the element-wise derivative of the map σ. Once $\nabla_{\boldsymbol{Z}} \mathcal{L}$ is obtained, $\nabla_{\boldsymbol{W}} \mathcal{L}$ and $\nabla_{\boldsymbol{\Omega}} \mathcal{L}$ are derived via the chain rule, eliminating the need to store activations at each layer during gradient updates.

However, IGNNs typically require multiple iterations to reach equilibrium, both in finding the fixed point during the forward process (2) and in solving the implicit differential during the backward process (5), resulting in significant overhead during both training and inference. Therefore, developing *a fast and efficient solver* to find their fixed points is crucial.

D. Traditional Anderson Acceleration Solver

Since implicit networks typically require multiple iterations to reach equilibrium, a commonly used solver method is Anderson Acceleration [16], which is widely applied in [3, 15, 17, 18]. Specifically, Given an implicit network layer f_θ and input $\boldsymbol{X}$, we aim to find the fixed-point $\boldsymbol{Z}^*$ in the system that represents the equilibrium state of the network output by solving a root-finding problem

$$g_\theta(\boldsymbol{Z}^\star, \boldsymbol{X}) := f_\theta(\boldsymbol{Z}^\star, \boldsymbol{X}) - \boldsymbol{Z}^\star = 0, \tag{6}$$

via utilizing fixed-point solvers, such as the classical Anderson Acceleration (AA) [16], which can directly seek the equilibrium point $\boldsymbol{Z}^\star$ through quasi-Newton methods. This approach demonstrates super-linear convergence properties.

We briefly introduce the AA-Solver, as our approach is inspired by it. Algorithm 1 provides a pseudocode example, where $\boldsymbol{G}^{[k]} = [g_\theta(\boldsymbol{Z}^{[k-m_k]}), \ldots, g_\theta(\boldsymbol{Z}^{[k]})]$ represents the past residuals. The key idea behind AA is to accelerate the next approximate solution by leveraging information stored from the history m iteration steps, constructing a normalized linear combination of past updates with $\boldsymbol{\alpha}^{[k]}$ and β. These weights are computed greedily to minimize the linear combination of past AA update steps. This approach is typically effective for accelerating the solution of implicit functions.

However, since the computation process relies only on the final output and does not require storing any intermediate activation values [17], the memory consumption during the training process remains constant, which is also a key reason why fixed-point solvers are attractive.

IV. Our proposed IGNN-Solver

Although traditional fixed-point solvers for IGNNs demonstrate functionality, they are characterized by relatively slow performance, necessitate manual parameter tuning, and are not well-suited for graph-based applications. To address these limitations, we propose a lightweight, learnable, and content-aware fixed-point solver for IGNN that incorporates a tiny graph neural network, combining the speed advantages of the solver with the information benefits of graph learning. By explicitly modeling the relationship between past residuals and the solver parameters $\boldsymbol{\alpha}^{[k]}$ and β, the proposed IGNN-Solver adaptively refines the convergence trajectory, leading to more effective and robust fixed-point updates, as illustrated in Figure 1.

Given the implicit model's characteristic that its representation capability is decoupled from the forward computation (i.e., the solver has no knowledge of the task, such as node classification, and vice versa), we can train this neural solver in a lightweight, unsupervised manner. Once learned, the IGNN-Solver can solve the fixed-point equation within the IGNN layer, facilitating further model updates. Note that our solver does not alter the original structure of the model but only accelerates the solution process. Details of the training strategy involving the IGNN-Solver are provided in Appendix C.

A. General Formulation

Let $\boldsymbol{X} \in \mathbb{R}^{d \times n}$ be the input features and $\widehat{\boldsymbol{A}} \in \mathbb{R}^{n \times n}$ be the graph, IGNN learns the node representation by finding the fixed point as detailed in (3). Algorithm 2 outlines the overall procedure of the IGNN-Solver. A learnable solver s_ξ is used to infer the parameters $\boldsymbol{\alpha}$ and β, where $\boldsymbol{\alpha}$ assigns

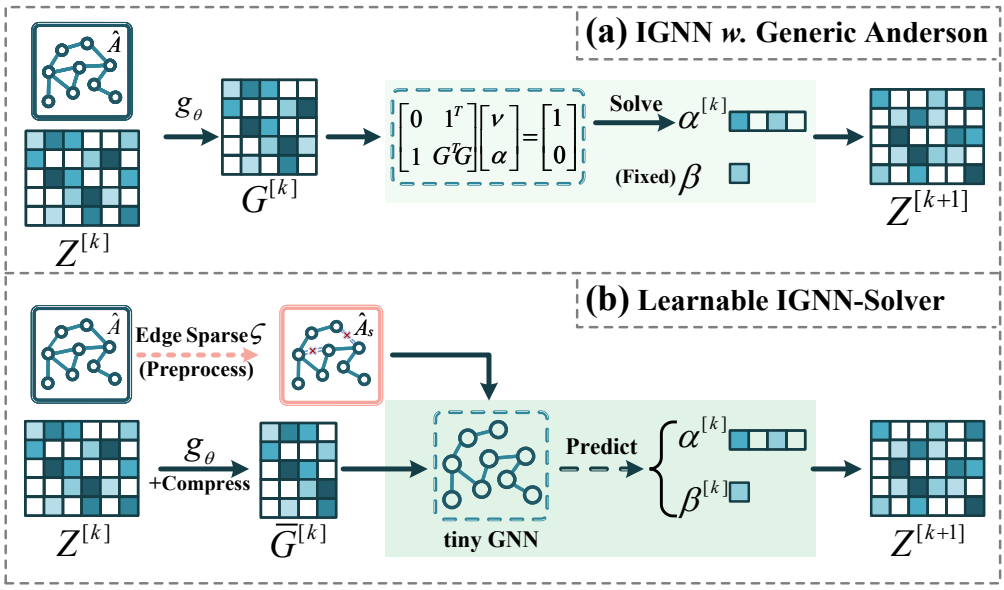


Fig. 1: Comparison of our IGNN-Solver and the generic Anderson solver at each iteration. (a) The canonical Anderson solver performs local least-squares fitting at each iteration, with a fixed $\beta = \beta^{[k]}$ set to a constant. (b) Our learnable IGNN-Solver improves and incorporates learnable iterative updates for enhanced convergence via tiny GNN.

Algorithm 2 IGNN-Solver Iterations and Training

Input: frozen IGNN model f_θ, initializer h_ϕ, tiny-GNN predictor s_ξ, sparse subgraph $\widehat{\boldsymbol{A}}_s$, storage $\boldsymbol{G} \in \mathbb{R}^{(m+1)\times d'}$.

Compute the initial value of fixed point by $\boldsymbol{Z}^{[0]} = h_\phi(\boldsymbol{X}) \in \mathbb{R}^{d'}$ ▷ initializer to guess initial value
Define $g_\theta(\boldsymbol{Z}) = f_\theta(\boldsymbol{Z}) - \boldsymbol{Z}$ and set $\boldsymbol{G}[0] = g_\theta(\boldsymbol{Z}^{[0]})$
for $k = 0, \ldots, K$ **do**
  1) Set $m_k = \min\{m, k\}$ and $\boldsymbol{G}^{[k]} = \boldsymbol{G}\,[0 : (m_k + 1)] \in \mathbb{R}^{(m_k+1)\times d'}$ ▷ compress storage
  2) Predict $\widehat{\boldsymbol{\alpha}}^{[k]}, \widehat{\beta}^{[k]} = s_\xi(\boldsymbol{G}^{[k]})$ ▷ predict with tiny-GNN
  3) Normalize coefficients: $\boldsymbol{\alpha}_i^{[k]} = \exp(\widehat{\boldsymbol{\alpha}}_i^{[k]}) / \sum_j \exp(\widehat{\boldsymbol{\alpha}}_j^{[k]})$, $\beta^{[k]} = (1 + \exp(-\widehat{\beta}^{[k]}))^{-1}$ ▷ normalization step
  4) Update by IGNN-Solver step: $\boldsymbol{Z}^{[k+1]} = \beta^{[k]} \cdot \mathbf{1}^\top \boldsymbol{G}^{[k]} + (1 - \beta^{[k]}) \sum_{i=0}^{m_k} \boldsymbol{\alpha}_i^{[k]} \boldsymbol{Z}^{[k-m_k+i]}$
  5) Update $\boldsymbol{G} = \text{concat}\left(\boldsymbol{G}[1:], [g_\theta(\boldsymbol{Z}^{[k+1]})]\right)$
end for
if it is inference stage **then**
  return $\boldsymbol{Z}^{[k+1]}$
else
  return $(\boldsymbol{Z}^{[k+1]}, \boldsymbol{G}^{[k]}, \boldsymbol{\alpha}^{[k]}, \beta^{[k]})_{k=0,\ldots,K}$ and $\boldsymbol{Z}^{[0]}$
  Compute $\mathcal{L}_{\text{total}}$ and back-propagate it to update IGNN-Solver $\{s_\xi, h_\phi\}$
end if

weights to the approximate solutions from the previous m iterations, and β modulates the combination of past residuals and approximate solutions. These parameters are subsequently applied to compute an accelerated update of Z via the *IGNN-Solver Step* defined in (11).

To train the IGNN-Solver, we minimize the joint loss function $\mathcal{L}_{\text{total}}$ by back-propagating through this K-step temporal process, which is discussed in Section IV-C. It's worth noting that the original IGNN is frozen (i.e., model parameters θ are fixed), and only the IGNN-Solver parameters ξ are trained here, so we do not need the ground-truth label that corresponds to input. This implies that IGNN-Solver can also be fine-tuned during inference after deployment.

1) Initializer: To accelerate the convergence of predicted values, we propose constructing an initializer

$$\boldsymbol{Z}^{[0]} = h_\phi(\boldsymbol{X}) \tag{7}$$

where $h_\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$, for a rapid and reasonable input-based initial estimate value, rather than simply setting the initial values to $\mathbf{0}$ or random, where ϕ are learnable parameters of the initializer. We set the intermediate dimension d' to be significantly smaller than the feature dimension d to reduce the training overhead of the initializer.

2) Improved Anderson Iterations with tiny-GNN: In the original AA solver, the parameter β is predetermined and fixed [16], whereas the parameters $\boldsymbol{\alpha}^{[k]}$s are determined by solving numerous linear equations using the least squares

method. Although this method operates adequately, its efficiency and adaptability in optimizing IGNN models are limited. Therefore, we propose introducing a tiny and learnable graph neural network as

$$\widehat{\boldsymbol{\alpha}}^{[k]}, \widehat{\beta}^{[k]} = s_\xi(\boldsymbol{G}^{[k]}, \widehat{\boldsymbol{A}}) \tag{8}$$

where $s_\xi : (\mathbb{R}^{(m_k+1)\times d'} \times \mathbb{R}^n) \to (\mathbb{R}^{(m_k+1)} \times \mathbb{R}^1)$, to predict the two parameters $\widehat{\boldsymbol{\alpha}}^{[k]}, \widehat{\beta}^{[k]}$ instead of setting them as the least squares solution on the past residuals $\boldsymbol{G}^{[k]}$.

To ensure that the accelerated iteration remains strictly within the class of mappings admitting monotonic convergence of Anderson iterations, we explicitly enforce the structural constraints prescribed by the theory. Specifically, we normalize the network outputs as

$$\boldsymbol{\alpha}_i^{[k]} = \mathrm{Softmax}(\widehat{\boldsymbol{\alpha}}^{[k]})_i = \frac{\exp(\widehat{\boldsymbol{\alpha}}_i^{[k]})}{\sum_{j=0}^{m_k} \exp(\widehat{\boldsymbol{\alpha}}_j^{[k]})}, \; i = 0, \ldots, m_k, \tag{9}$$

$$\beta^{[k]} = \mathrm{Sigmoid}(\hat{\beta}^{[k]}) = \frac{1}{1 + e^{-\hat{\beta}^{[k]}}} \tag{10}$$

which enforce the convexity constrains $\boldsymbol{\alpha}_i^{[k]} \geq 0$ and $\sum_i^{m_k} \boldsymbol{\alpha}_i^{[k]} = 1$, together with the boundedness constraint $\beta^{[k]} \in (0, 1]$.

Incorporating these learnable parameters into the classical Anderson scheme yields the proposed IGNN-Solver, whose update rule is given by

$$\begin{aligned} \boldsymbol{Z}^{[k+1]} = \beta^{[k]} \sum_{i=0}^{m_k} \boldsymbol{\alpha}_i^{[k]} f_\theta(\boldsymbol{Z}^{[k-m_k+i]}) \\ + (1 - \beta^{[k]}) \sum_{i=0}^{m_k} \boldsymbol{\alpha}_i^{[k]} \boldsymbol{Z}^{[k-m_k+i]}, \end{aligned} \tag{11}$$

where $f_\theta(\cdot)$ denotes the IGNN model defined in (3) with the weight matrix $\boldsymbol{W}$ reparameterized via the Cayley transform in (4), and the parameters $\boldsymbol{\alpha}^{[k]}$ and $\beta^{[k]}$ are defined in (9)–(10), respectively. Notably, the Cayley reparameterization guarantees the well-posedness of the IGNN model, while the constraints in (9)–(10) ensure that each accelerated update is a convex combination of past iterates followed by a contractive relaxation, thereby satisfying the assumptions underpinning monotonic convergence analysis.

The well-posedness and the convergence of IGNN-Solver are established in the following theorem.

Theorem 1 (Well-posedness and Convergence of IGNN–Solver)**.** Consider an IGNN model defined in (3) whose weight matrix $\boldsymbol{W}$ is reparameterized via the Cayley transform in (4), and let $\boldsymbol{Z}^{[k]}$ denote the k-th iterate generated by the IGNN-Solver update defined in (11) whose coefficients $\boldsymbol{\alpha}^{[k]}$ and $\beta^{[k]}$ are constrained as specified in (9)–(10). Then the following statements hold

(i) The IGNN model is well-posed, i.e., the unique fixed point $\boldsymbol{Z}^\star$ of (3) exists;

(ii) For all $k \geq 0$, the iterates satisfy a monotonic error contraction relative to the history window, i.e.,

$$\|\boldsymbol{Z}^{[k+1]} - \boldsymbol{Z}^\star\|_F \leq \gamma_k \cdot \max_{0 \leq j \leq m_k} \|\boldsymbol{Z}^{[k-j]} - \boldsymbol{Z}^\star\|_F$$

with $0 < \gamma_k < 1$. Consequently, the sequence $\{\boldsymbol{Z}^{[k]}\}_{k\geq 0}$ converges globally to $\boldsymbol{Z}^\star$.

Remark 1. The proposed IGNN-Solver retains the foundational characteristics of classical Anderson acceleration. By enforcing the Cayley reparameterization of the weight matrix (4) and normalizing the parameters $\boldsymbol{\alpha}^{[k]}$ and $\beta^{[k]}$ within their admissible ranges (9)–(10), the solver maintains the contractive properties required for monotonic convergence. The complete proof is deferred to Appendix A.

B. Graph Sparsification and Storage Compression

Directly using the original graph for modeling would *result in an oversized network* [19, 20], deviating from the initial goal of using a tiny network for prediction. Moreover, because of *the curse of dimensionality*, mapping the data from high to extremely low dimensions is not appropriate [21]. Therefore, in response to the challenges posed by large-scale graphs, we take into account both *Graph Sparsification* and *Storage Compression* in the design of the solver, as detailed below.

1) Graph Sparsification: Firstly, the number of edges in graph $\widehat{\boldsymbol{A}}$ is usually large in practice, as it is affected by the scale of the input (for example, in the ogbn-products dataset, the number of edges in the graph is close to 62M). To maintain the lightweight nature of tiny-GNN and reduce the computational cost of prediction, we propose introducing graph sparsification [53–55] for a light graph:

$$\widehat{\boldsymbol{A}}_s = \varsigma(\widehat{\boldsymbol{A}}, \beta), \tag{12}$$

where $\widehat{\boldsymbol{A}}_s$ represents the sparse subgraph of $\widehat{\boldsymbol{A}}$, obtained via a graph sparsification operator ς, with the parameter β controls the sparsity of $\widehat{\boldsymbol{A}}_s$.

We adopt the RPI-Graph [19] framework as our graph sparsification strategy due to its task-agnostic nature and theoretical rigor. Specifically, it is a plug-and-play method rooted in the Principle of Relevant Information (PRI), which identifies a subgraph that minimizes divergence from the original graph's spectral properties while satisfying a pre-defined edge budget. Crucially, we apply this sparsification strategy exclusively to large-scale datasets to optimize computational efficiency where initialization overhead is most significant.

In our implementation, we turn the sparsification ratio for the large-scale benchmarks via the hyperparameter β_{sp}. It is worth noting that this sparse subgraph is utilized solely within the auxiliary tiny-GNN predictor s_ξ to generate parameters. Consequently, this design ensures that the substantial reduction in parameter predictor s_ξ overhead is achieved, while the core IGNN model f_θ predicts its representation by operating on the original, full-rank graph. This decoupling maintains the high-fidelity representation typical of implicit models without structural information loss during the final equilibrium state calculation. A detailed comparison of graph statistics (including the sparsification ratio ρ_{sp}, the hyperparameter β_{sp}, and the degree distribution shift characterizing the changes in the sparsified graphs compared with the original graphs) is provided in Table VIII, Appendix B-C.

2) Storage Compression: Furthermore, given that the number of nodes n is relatively large (e.g., n is approximately 169K in the ogbn-arxiv dataset), directly mapping s_ξ from a high-dimensional space to a significantly lower-dimensional one in (8) is suboptimal [36, 56, 57] due to the curse of dimensionality [21]. Therefore, to ensure s_ξ remains both computationally efficient and compact, we propose compressing $\boldsymbol{G}_i^{[k]}$ into a smaller yet representative form, denoted as $\bar{\boldsymbol{G}}_i^{[k]}$. Specifically, we employ a MLP to project the features from $\mathbb{R}^{(m_k+1)\times d'}$ to a target space $\mathbb{R}^{(m_k+1)\times p}$. Subsequently, the m_k+1 most recent feature matrices are concatenated:

$$\bar{\boldsymbol{G}}^{[k]} = \mathop{\Big\|}_{i=k-m_k}^{k} \mathrm{MLP}[\boldsymbol{G}_i^{[k]}], \tag{13}$$

where $||_{i=k-m_k}^{k}$ denotes the concatenation operation that stacks the compressed matrices $\bar{\boldsymbol{G}}_i^{[k]} \in \mathbb{R}^{(m_k+1)\times p}$ from the current and previous m_k iterations.

Once the smaller yet representative storage matrix $\bar{\boldsymbol{G}}$ is obtained as described above, we treat it as a p-channel matrix and employ a tiny graph neural network layer:

$$\boldsymbol{\alpha}^{[k]}, \beta^{[k]} = s_\xi(\bar{\boldsymbol{G}}^{[k]}, \widehat{\boldsymbol{A}}_s), \tag{14}$$

to predict the relative weights $\boldsymbol{\alpha}^{[k]}$ assigned to past m_k+1 residual, as well as the AA mixing coefficient $\beta^{[k]}$ at the k-th iteration.

In this way, s_ξ shall autonomously learn and adjust these parameters $\boldsymbol{\alpha}^{[k]}$ and $\beta^{[k]}$ based on previous solver steps, while also receiving gradients from subsequent iterations. We provide a comparison of different choices for s_ξ in Section V-F, demonstrating that IGNN-Solver improves the convergence path. We also introduce the training procedure below.

C. Training IGNN-Solver

Unlike the neural ODE solver, the fixed-point trajectory of the IGNN is not unique. Therefore, trajectory fitting is not applicable to the IGNN-solver training. Instead, the goal is simply to bring everything as close as possible to $\boldsymbol{Z}^\star$. Formally, given an IGNN-Solver $\{s_\xi, h_\phi\}$ that returns $(\boldsymbol{Z}^{[k+1]}, \boldsymbol{G}^{[k]}, \boldsymbol{\alpha}^{[k]}, \beta^{[k]})_{k=0,\dots,K}$ and $\boldsymbol{Z}^{[0]}$, we introduce three objectives functions for its training. The complete training algorithm can be referred to in Algorithm 2.

a) Initializer Loss Functions: To train the initializer, we minimize the distance between the initial guess and the fixed-point by

$$\mathcal{L}_{\text{init}} = \left\| h_\phi(\boldsymbol{X}) - \boldsymbol{Z}^\star \right\|_F, \tag{15}$$

for facilitating the subsequent iteration process. Since the initializer directly predicts based on the input $\boldsymbol{X}$ without going through iteration, we separate this loss from other components.

b) Reconstruction Loss Functions: The training of the solver does not require reference to label information or any trajectory fitting. Apart from the loss $\mathcal{L}_{\text{init}}$ necessary for training the initializer above, we introduce reconstruction loss

$$\mathcal{L}_{\text{rec}}^{[k]} = \left\| \boldsymbol{Z}^{[k]} - \boldsymbol{Z}^\star \right\|_F, \tag{16}$$

where the reconstruction loss $\mathcal{L}_{\text{rec}}^{[k]}$ aims to make all intermediate predictions $\boldsymbol{Z}^{[k]}$ converge to the accurate fixed point $\boldsymbol{Z}^\star$ as closely as possible.

c) Auxiliary Loss Functions: Although we utilize $\boldsymbol{\alpha}^{[k]}$ and $\beta^{[k]}$ to improve the generic solver, we have empirically found that using an auxiliary loss to guide the solver's prediction of $\boldsymbol{\alpha}$ is beneficial sometimes, especially in the early stages of training. Therefore, in practice, we suggest considering this loss and gradually diminishing it as training progresses (i.e., decay the weight of this loss to 0).

$$\mathcal{L}_{\boldsymbol{\alpha}} = \sum_{k=0}^{K} \left\| \boldsymbol{G}^{[k]} \boldsymbol{\alpha}^{[k]} \right\|_F, \tag{17}$$

The final form of the joint loss function is as follows:

$$\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{rec}}^{[k]} + \lambda_2 \mathcal{L}_{\text{init}} + \lambda_3 \mathcal{L}_{\boldsymbol{\alpha}}, \tag{18}$$

where λ_1, λ_2 and λ_3 control the weights of three loss functions mentioned above and elaborate more in Appendix B-B.

V. Experiments

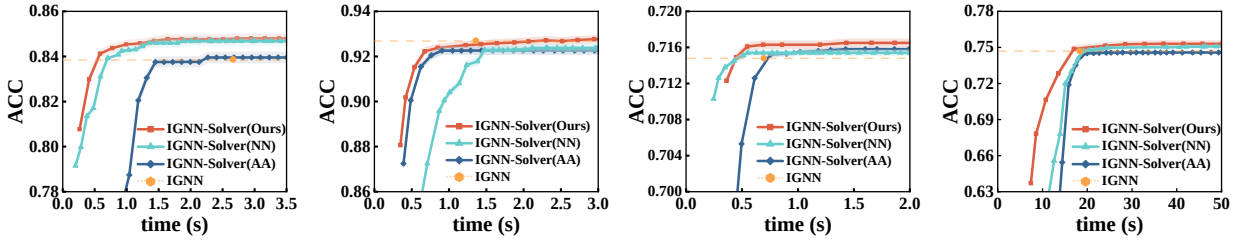
In this section, we compare the speed/accuracy Pareto curve rather than a single point on the curve of IGNN-Solver with IGNNs and several state-of-the-art (SOTA) GNNs across various graph classification tasks, both at the node and graph levels. Our goal is to demonstrate: 1) IGNN-Solver achieves nearly a $1.5\times$ to $8\times$ inference acceleration without sacrificing accuracy; and 2) IGNN-Solver is extremely compact, adding little overhead to training. Specifically, we compare our approach with numerous representative baselines and two variants of our method: IGNN *w.* AA (using the original Anderson Acceleration to speed up IGNN) and IGNN *w.* NN (replacing our proposed graph neural solver with a standard neural network solver for parameter learning).

We also evaluate IGNN-Solver on graph classification tasks to further demonstrate the effectiveness and versatility. We sequentially demonstrate that the training overhead on IGNN-Solver is little relative to the total time overhead on IGNN. We conduct detailed cost and amortization analyses to quantify training and inference overheads, complemented by an efficiency study that examines convergence behavior, per-epoch runtime, memory consumption, and training dynamics. We conduct ablation studies to validate the stability of the IGNN-Solver and assess the effectiveness of its components. Additional experimental setups and descriptions are detailed in Appendix B.

Furthermore, we comprehensively evaluate IGNN-Solver on multi-label, heterogeneous, and dynamic graph benchmarks, as well as on recently popular hypergraph benchmarks, with the corresponding results and discussions detailed in Appendix E.

A. Experiments Settings

1) Datasets: The evaluation is conducted on different-field, real-world benchmarks for node classification tasks, including four citation datasets (Citeseer, ACM, CoraFull, ogbn-arxiv), three social interaction datasets (BlogCatalog, Flickr, Reddit), and two product network datasets (Amazon-all, ogbn-products). Notably, to demonstrate the scalability of the proposed model on larger datasets, we use four *large-scale datasets*: Amazon-all, Reddit, ogbn-arxiv, and ogbn-products, two of which are from the Open Graph Benchmark



(a) Amazon-all (b) Reddit (c) ogbn-arxiv (d) ogbn-products

Fig. 2: Speed-accuracy tradeoff curves with error bars comparing the inference time of IGNN with IGNN-Solver and its variants IGNN-Solver (NN) and IGNN-Solver (AA), across four *large-scale* datasets: Amazon-all (2a), Reddit (2b), ogbn-arxiv (2c) and ogbn-products (2d). The x-axis represents the inference time, while the y-axis denotes the model accuracy. Each plot shows the average performance over five independent runs under identical experimental conditions.

TABLE I: Node classification accuracy results on four *large-scale* real-world datasets (more results on small-scale datasets including Citeseer, ACM, CoraFull, BlogCatalog, Flickr can be found in Table III), with experiments conducted over five trials. The mean accuracy (%) ± standard deviation is reported. The **best** and the runner-up results are highlighted in boldface and underline, respectively.

Type	*Models *Nodes *Edges	Amazon-all 334, 863 14, 202, 057	Reddit 232, 965 11, 606, 919	ogbn-arxiv 169, 343 1, 166, 243	ogbn-products 2, 449, 029 61, 859, 140
Explicit	GCN [12]	79.12 ± 1.11	89.65 ± 0.14	71.56 ± 0.25	71.91 ± 0.27
	GAT [34]	76.02 ± 2.09	90.08 ± 0.44	71.10 ± 0.24	72.33 ± 0.32
	SGC [58]	75.56 ± 1.75	91.44 ± 0.41	64.66 ± 1.02	70.48 ± 0.19
	APPNP [59]	79.80 ± 1.22	91.68 ± 0.24	71.28 ± 0.29	74.46 ± 0.64
	JKNet [60]	81.19 ± 1.09	91.71 ± 0.31	71.08 ± 0.35	73.70 ± 0.58
	AM-GCN [36]	81.85 ± 1.46	90.20 ± 0.34	68.83 ± 0.67	73.19 ± 0.49
	DEMO-Net [28]	84.08 ± 1.29	89.53 ± 0.29	68.40 ± 0.24	73.88 ± 0.62
	GCNII [61]	83.60 ± 2.40	89.87 ± 0.38	67.66 ± 0.20	72.88 ± 0.18
	ACM-GCN [62]	83.04 ± 2.61	90.37 ± 0.42	68.32 ± 0.18	71.68 ± 0.41
Implicit	IGNN [1]	83.90 ± 0.51	92.30 ± 1.55	70.49 ± 0.75	74.63 ± 0.24
	EIGNN [2]	$\underline{84.32} \pm 0.57$	92.00 ± 0.24	70.59 ± 0.31	74.58 ± 0.26
	MIGNN [3]	83.68 ± 0.82	91.98 ± 0.42	$\underline{71.95} \pm 0.44$	74.62 ± 0.32
	MGNNI [13]	83.14 ± 0.90	$\underline{92.66} \pm 0.17$	71.20 ± 0.40	OOM
	SEIGNN [14]	84.06 ± 0.70	92.48 ± 0.11	70.75 ± 0.46	74.10 ± 0.21
	IGNN-Solver (AA)	83.44 ± 0.19	92.37 ± 0.35	71.73 ± 0.41	74.61 ± 0.28
	IGNN-Solver (NN)	84.13 ± 1.04	92.42 ± 0.41	70.78 ± 0.37	$\underline{74.69} \pm 0.31$
	IGNN-Solver (Ours)	$\mathbf{84.50} \pm 0.70$	$\mathbf{93.91} \pm 0.31$	$\mathbf{72.53} \pm 0.41$	$\mathbf{74.90} \pm 0.20$

(OGB) [24]. In particular, we include experiments on the `ogbn-papers100M` dataset which contains approximately 111M nodes and 1.6B edges. To our best knowledge, this is the first work to successfully apply and evaluate IGNNs on this billion-scale benchmark. In addition, we use five bioinformatics benchmarks (MUTAG, PTC_MR, COX2, PROTEINS, NCI1) for graph classification tasks. For further details, please refer to Appendix B.

2) Baselines: For graph node classification tasks, we compare IGNN-Solver with a comprehensive set of baselines, including three implicit GNNs, i.e., IGNN [1], EIGNN [2], MIGNN [3], MGNNI [13], SEIGNN [14]; and nine explicit/traditional GNNs, i.e., AM-GCN [36], GCN [12], GAT [34], SGC [58], APPNP [59], JKNet [60], DEMO-Net [28], GCNII [61], ACM-GCN [62]. In addition, for ultra-scale dataset `ogbn-papers100M`, we compare IGNN-Solver against with MLP [58], SGC [58], GraphSAGE [23], Node2vec [63], IGNN [1], and GLEM+GIANT+GAMLP [64] (which is currently the highest reported test accuracy on the official OGB leaderboard). For graph classification tasks, we compare seven explicit models, i.e., WL [65], DCNN [66], DGCNN [67], GIN [68], FDGNN [69], PK [70], and GCN [12], against five implicit graph models, including IGNN [1], CGS [71], GIND [72], MIGNN [3], and our proposed IGNN-Solver.

For the sake of fairness, we use the same hidden layer configuration for all baseline models. For example, in the Flickr dataset, the dimension of the hidden layers is uniformly set to 512 for all methods (including ours, see in Table VI). In addition, for EIGNN, the arbitrary gamma is set to 0.8, which is consistent with the original paper. For AM-GCN, the number of nearest neighbors in the KNN graph is set from $5, 6, 7$. For GCN, the hidden layer setup is the same as that for others. For GAT, three attention headers are used for each layer. For SGC, the power of self-loops in the graph adjacency matrix is set to 3. For APPNP, we set the number of iterations to 3 and teleport probability $\boldsymbol{\alpha}_{\text{APPNP}}$ to 0.5. For JKNet, we set layers to 1 in small datasets and set to 8 in large ones. For DEMO-Net, the regularization parameter is set as $5e-4$, the learning rate is set as $5e-3$, and the hash dimension is set as 256. For GCNII, we set $\boldsymbol{\alpha}_\ell = 0.1$ and L_2

TABLE II: Node classification accuracy on `ogbn-papers100M` (with 1.6B edges and 111M nodes). The mean accuracy ± standard deviation of test accuracy (%), inference time (s) and parameters are reported. The **best** and the runner-up results are highlighted in boldface and underline, respectively. Inference time is only reported for implicit models.

Type	Method	Test Accuracy (%)	Inference Time (s)	# Params
Explicit	MLP	47.24 ± 0.31	-	1.16M
	SGC	63.29 ± 0.19	-	0.14M
	GraphSAGE	62.82 ± 0.01	-	5.76M
	Node2vec	55.60 ± 0.23	-	14.22B
	GLEM+GIANT+GAMLP*	**70.37 ± 0.02**	-	154.78M
Implicit	IGNN	65.11 ± 0.55	227.69	1.49M
	IGNN-Solver (AA)	65.07 ± 0.71	181.48	1.49M
	IGNN-Solver	63.06 ± 0.23	36.24	1.50M

regularization to $5e-4$, consistent with the original paper. For ACM-GCN, we set layers to 1 in small datasets and set to 4 in large ones. For our IGNN-Solver and its variants, we employ the same settings as the basic IGNN model. Additionally, for graph classification tasks, performances for explicit baselines (WL [65], DCNN [66], DGCNN [67], GIN [68], FDGNN [69], PK [70], and GCN [12]) are taken from [3]; as well as implicit baselines (IGNN [1], CGS [71], GIND [72], MIGNN [3]) are taken from [72]. Furthermore, we adjust these parameters affecting the convergence and performance of IGNN through a combined approach of hierarchical grid search and manual tuning. The Adam optimizer [73] is used for optimization.

B. Experiments on Node Classification Tasks

1) Experimental setup: To demonstrate the superiority of the IGNN-Solver over IGNNs in terms of both performance and efficiency, we analyze the movement of the speed/accuracy Pareto curve across various datasets, rather than concentrating on a single point on the curve, which is depicted in Figures 2 and 4. All experiments are conducted five times, and the best results are reported. Furthermore, we present the node classification performance results for each method on various datasets, which are shown in Tables I and III. All experiments are conducted five times, and we report the average results along with the standard deviation. Additional details on the training procedure and the hyperparameters used for each task are provided in Appendices B and C.

2) Results on large-scale tasks: In Figure 2, we present the speed/accuracy Pareto curves on large-scale datasets, including Amazon-all, Reddit, ogbn-arxiv, and ogbn-products. The curves are obtained by varying the number of iterations, which acts as the primary control parameter for trading off computational cost against convergence accuracy. The standard IGNN baseline is shown as a single reference point in each plot. By drawing a horizontal line through this baseline, we can clearly compare the inference time and accuracy achieved by the proposed IGNN-Solver under equivalent performance levels. These results reveal several key findings:

1) *Comprehensive performance and efficiency*: IGNN-Solver generally outperforms all other methods, particularly in large datasets with a greater graph radius. This advantage is more pronounced in such cases. The improved performance is attributed to IGNN-Solver's significant enhancement in the convergence path and the better initialization of the fixed-point equation through the initializer;
2) *Rapid inference advantage*: IGNN-Solver demonstrates a notable speed advantage during inference. For instance, on the large-scale dataset Amazon-all, our IGNN-Solver achieves $8\times$ faster inference speed than IGNN while maintaining the same performance, and there is consistently at least $1.5\times$ acceleration and a similar pattern has been observed as well on other datasets.
3) *Fundamental innovation in solver dynamics*: Beyond speed and accuracy gains, the results in Figure 2 highlight that IGNN-Solver explicitly targets the *solver trajectory*, in contrast to previous works that focus on redesigning the IGNN formulation. By learning to adapt the convergence path across iterations, our method reshapes the trajectory of the fixed-point solver rather than modifying the fixed-point equation. This trajectory-centric perspective enables a more efficient traversal of the solution space, leading to faster convergence under equivalent accuracy constraints.

From Table I, we observe that implicit GNNs with infinite depth generally outperform shallow explicit GNNs in most cases. Furthermore, IGNN-Solver consistently shows higher accuracy percentages across most datasets compared to other state-of-the-art explicit GNNs like DEMO-Net [28], GCNII [61] etc, and advanced implicit GNNs like MGNNI [13] and SEIGNN [14].

To further evaluate the scalability of IGNN-Solver, we conduct experiments on an extremely large graph, namely the `ogbn-papers100M` dataset, which comprises 111 M nodes and 1.6 B edges [24]. To ensure a fair evaluation, we adopt the same data splitting and preprocessing protocol as GraphSAGE [23]. Table II summarizes the node classification accuracy, inference efficiency, and computational configurations of all baseline methods and our proposed model on the challenging `ogbn-papers100M` graph, while Figure 3 presents the speed/accuracy Pareto curves. The results demonstrate that our approach scales effectively to billion-edge graphs, achieving the most favorable inference efficiency across all compared methods while maintaining comparable performance.

3) Results on small-scale tasks: We further evaluate IGNN-Solver on several small-scale graph node classification tasks, including Citeseer, ACM, CoraFull, BlogCatalog, and Flickr. The hyperparameters used are outlined in Table VI, Appendix B-B. The mean accuracy and standard deviation are

TABLE III: Node Classification accuracy results on five *small-scale* real-world datasets, with experiments conducted over five trials. The mean accuracy (%) ± standard deviation is reported. The **best** and the runner-up results are highlighted in boldface and underline, respectively.

Type	*Models *Nodes *Edges	Citeseer 3,327 4,732	ACM 3,025 13,128	CoraFull 19,793 65,311	BlogCatalog 5,196 171,743	Flickr 7,575 239,738
Explicit	GCN [12]	64.80 ± 4.15	83.78 ± 3.95	35.98 ± 9.44	71.30 ± 1.12	76.98 ± 1.83
	GAT [34]	71.24 ± 2.88	80.60 ± 5.12	50.24 ± 1.87	76.24 ± 2.76	72.64 ± 3.23
	SGC [58]	70.32 ± 2.75	85.40 ± 1.23	46.56 ± 4.43	69.76 ± 1.11	71.88 ± 3.47
	APPNP [59]	61.56 ± 8.92	84.16 ± 4.25	21.24 ± 4.76	61.34 ± 9.39	73.42 ± 3.80
	JKNet [60]	63.78 ± 8.76	64.96 ± 5.47	23.04 ± 6.01	72.62 ± 6.83	75.68 ± 1.15
	AM-GCN [36]	73.10 ± 1.62	89.56 ± 0.30	53.40 ± 1.59	73.86 ± 1.10	76.86 ± 2.02
	DEMO-Net [28]	68.34 ± 2.94	84.38 ± 2.19	61.74 ± 3.65	74.26 ± 2.70	75.60 ± 3.95
	GCNII [61]	71.98 ± 0.80	85.36 ± 1.05	57.64 ± 3.34	74.94 ± 3.81	79.92 ± 2.14
	ACM-GCN [62]	72.38 ± 1.46	88.98 ± 0.41	59.88 ± 1.59	**78.18** ± 1.75	74.82 ± 3.78
Implicit	IGNN [1]	72.96 ± 1.83	90.88 ± 0.95	65.52 ± 0.51	75.68 ± 0.55	75.80 ± 0.29
	EIGNN [2]	72.38 ± 1.36	88.36 ± 1.03	61.80 ± 0.60	75.34 ± 0.38	75.66 ± 0.94
	MIGNN [3]	73.79 ± 0.94	89.59 ± 1.61	62.94 ± 0.46	76.68 ± 1.49	74.96 ± 0.49
	MGNNI [13]	73.21 ± 0.72	89.24 ± 0.53	62.57 ± 1.89	76.97 ± 0.41	75.94 ± 0.59
	SEIGNN [14]	73.20 ± 0.32	90.18 ± 0.07	65.49 ± 0.66	77.16 ± 0.79	76.71 ± 1.17
	IGNN-Solver (AA)	75.28 ± 0.38	**91.34** ± 0.46	65.88 ± 0.34	76.82 ± 0.34	75.84 ± 0.27
	IGNN-Solver (NN)	74.78 ± 0.42	91.08 ± 0.56	65.62 ± 0.16	76.88 ± 0.74	78.55 ± 0.49
	IGNN-Solver (Ours)	**75.60** ± 0.22	91.20 ± 1.15	**66.08** ± 1.23	77.64 ± 0.54	**80.14** ± 0.23

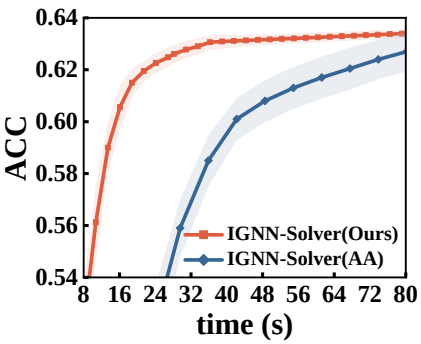


Fig. 3: Speed-accuracy tradeoff curves with error bars comparing the inference time of **IGNN-Solver** with its variants **IGNN-Solver (AA)** on ultra-scale dataset `ogbn-papers100M`. The x-axis represents the inference time, while the y-axis denotes the model accuracy. Each plot shows the average performance over five independent runs under identical experimental conditions.

reported in Table III, while the speed/accuracy Pareto curve is presented in Figure 4. In general, learning LRD is not crucial for these tasks, as the graph diameters are relatively small [3]. However, as shown in Table III, even for these small-scale node classification tasks, IGNN-Solver outperforms IGNNs and many explicit GNNs, as well as other advanced implicit models. As illustrated in Figure 4, the solver continues to improve the convergence path of IGNN, even in small-scale tasks. These results, along with those in Section V-B, confirm the expressive power of the IGNN-Solver using learnable graph neural networks, even surpassing that of many explicit and enhanced implicit GNNs.

C. Experiments on Graph Classification Tasks

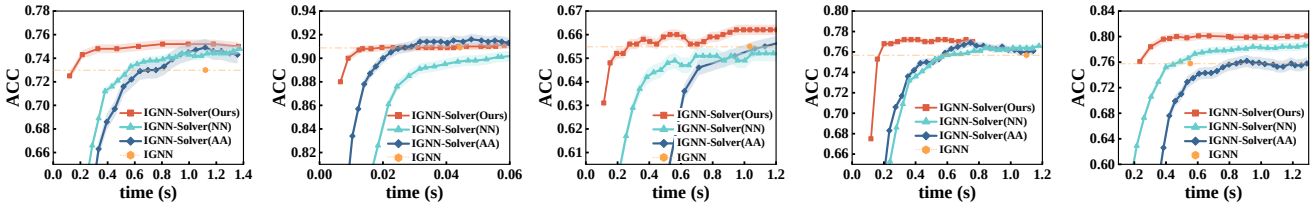
1) Experimental setup: We extend our evaluation to graph classification tasks to further demonstrate the effectiveness and versatility of IGNN-Solver. We conduct experiments on several widely used bioinformatics benchmark datasets for graph classification, including MUTAG, PTC_MR, COX2, PROTEINS, and NCI1. These datasets vary in size and complexity, as detailed in Appendix B. We train the model using 10-fold cross-validation using the experimental setup of [1]. For each dataset, we employ standard 10-fold cross-validation to evaluate the test performance of the IGNN-Solver, with the baseline and their results drawn from [3, 72]. The average prediction accuracy and standard deviations are in Table IV.

2) Results and discussion: The results of the graph classification experiments, summarized in Table IV, show that our IGNN-Solver almost outperforms the baseline models across all datasets, clearly demonstrating its ability to effectively capture graph structures and significantly improve classification accuracy. These findings confirm the effectiveness of the IGNN-Solver in graph classification tasks, further supporting its potential for deployment in real-world applications where graph data is prevalent.

D. Efficiency study

We present additional evidence on the convergence and generalizability of the neural solvers in Figure 5, where the ordinate represents the relative residual $\frac{||f(z)-z||_F}{||z||_F}$. We compare the convergence of a pre-trained IGNN model under two conditions: 1) canonical iteration in IGNN [1], and 2) IGNN-Solver with $K = 6$ unrolling steps.

From Figure 5, we observe that the IGNN-Solver, trained with K unrolling steps, is capable of generalizing beyond K, with both solvers ultimately reaching a stable state, demonstrating good convergence. Moreover, thanks to the fixed-point solution of the graph neural parameterization, IGNN-Solver continuously enhances the convergence of the canonical iterators while being more computationally efficient. Specifically, each step of the neural solver is cheaper than the

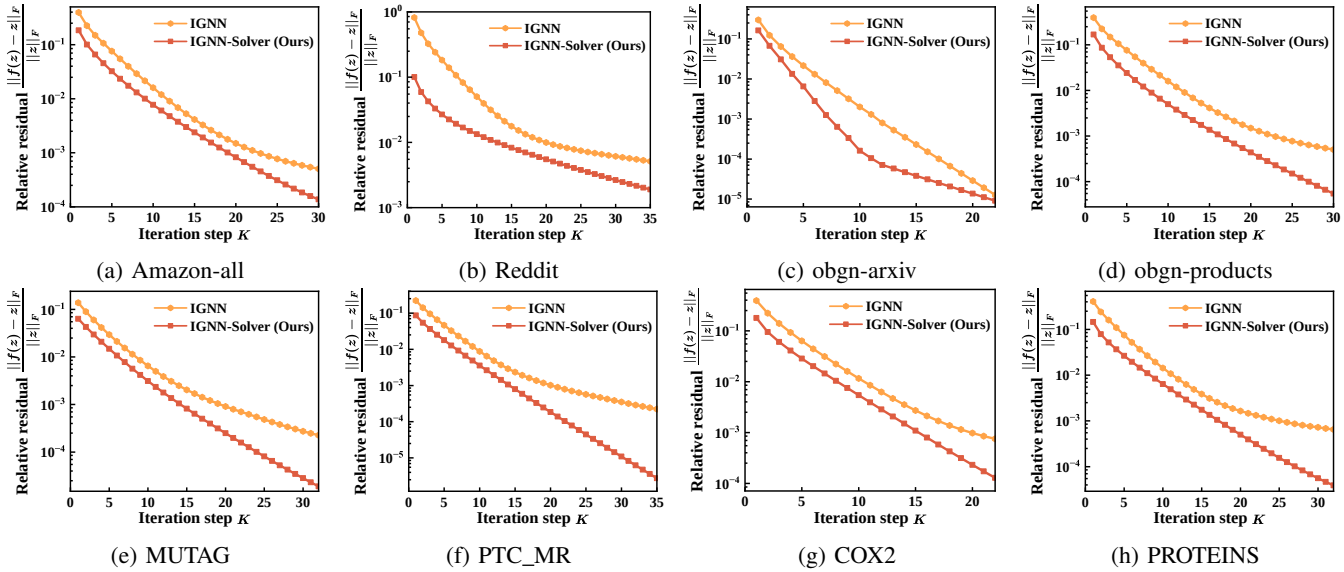


(a) Citeseer (b) ACM (c) CoraFull (d) BlogCatalog (e) Flickr

Fig. 4: Speed-accuracy tradeoff curves with error bars comparing the inference time of IGNN with IGNN-Solver and its variants IGNN-Solver (NN) and IGNN-Solver (AA), across five *small-scale* datasets: Citeseer (4a), ACM (4b), CoraFull (4c), BlogCatalog (4d) and Flickr (4e). The x-axis represents the inference time, while the y-axis denotes the model accuracy. Each plot shows the average performance over five independent runs under identical experimental conditions.

TABLE IV: Graph classification accuracy results on five benchmark datasets. The mean accuracy (%) ± standard deviation is reported. The **best** and the runner-up results are highlighted in boldface and underline, respectively.

Type	Method	MUTAG	PTC_MR	COX2	PROTEINS	NCI1
Explicit	WL [65]	84.1 ± 1.9	58.0 ± 2.5	83.2 ± 0.2	74.7 ± 0.5	**84.5 ± 0.5**
	DCNN [66]	67.0	56.6	-	61.3	62.6
	DGCNN [67]	85.8	58.6	-	75.5	74.4
	GIN [68]	89.4 ± 5.6	64.6 ± 7.0	-	76.2 ± 2.8	82.7 ± 1.7
	FDGNN [69]	88.5 ± 3.8	63.4 ± 5.4	83.3 ± 2.9	76.8 ± 2.9	77.8 ± 1.6
	PK [70]	76.0 ± 2.7	59.5 ± 2.4	81.0 ± 0.2	73.7 ± 0.7	82.5 ± 0.5
	GCN [12]	85.6 ± 5.8	64.2 ± 4.3	-	76.0 ± 3.2	80.2 ± 2.0
Implicit	IGNN [1]	76.0 ± 13.4	60.5 ± 6.4	79.7 ± 3.4	76.5 ± 3.4	73.5 ± 1.9
	CGS [71]	89.4 ± 5.6	64.7 ± 6.4	-	76.3 ± 4.9	77.6 ± 2.0
	GIND [72]	89.3 ± 7.4	66.9 ± 6.6	84.8 ± 4.2	77.2 ± 2.9	78.8 ± 1.7
	MIGNN [3]	81.8 ± 9.1	**72.6 ± 5.4**	85.0 ± 5.3	77.9 ± 2.6	79.6 ± 1.8
	IGNN-Solver (Ours)	**90.5 ± 1.6**	71.8 ± 3.1	**85.3 ± 3.0**	**78.4 ± 2.6**	80.2 ± 1.1



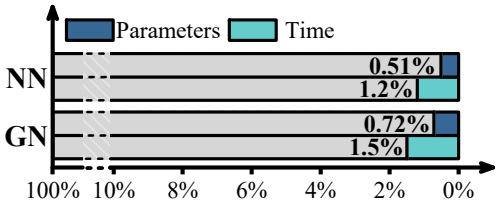
(a) Amazon-all (b) Reddit (c) obgn-arxiv (d) obgn-products

(e) MUTAG (f) PTC_MR (g) COX2 (h) PROTEINS

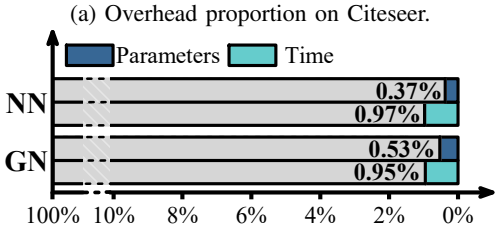
Fig. 5: Comparison of convergence curves across eight datasets, highlighting the substantial acceleration in convergence achieved by our IGNN-Solver.

TABLE V: Amortization cost on large-scale datasets. We compare per-epoch training/inference time and peak GPU memory usage across various benchmarks. The results demonstrate that our method achieves a 3.7–6.3× inference speedup with negligible training overhead (∼0.04×) and comparable memory cost.

Dataset	Models	Train Time (s/epoch)	Inf. Time (s/epoch)	Speedup	Peak GPU Mem. (GB)
ogbn-arxiv	IGNN (Picard)	1.25	0.70	1.00×	6.28
	IGNN-Solver (AA)	1.25	0.38	1.84×	6.54
	IGNN-Solver (Ours)	1.37 (↑8.8%)	**0.19**	**3.68**×	6.83
ogbn-products	IGNN (Picard)	84.20	18.32	1.00×	13.56
	IGNN-Solver (AA)	84.35	4.85	3.78×	13.80
	IGNN-Solver (Ours)	88.51 (↑5.1%)	**4.01**	**4.57**×	15.52
ogbn-papers100M	IGNN (Picard)	1,245.3	227.69	1.00×	12.26
	IGNN-Solver (AA)	1,248.0	181.48	1.25×	12.62
	IGNN-Solver (Ours)	1,296.8 (↑4.0%)	**36.24**	**6.28**×	13.44

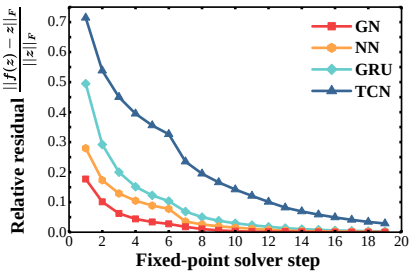


(a) Overhead proportion on Citeseer.

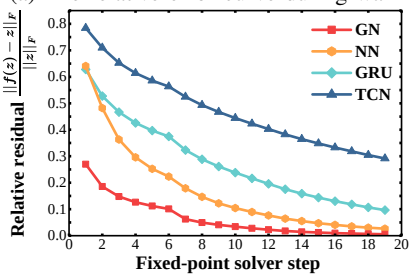


(b) Overhead proportion on Amazon-all.

Fig. 6: Relative parameter size and training time of the IGNN-Solver on the small Citeseer dataset and the large Amazon-all dataset, with consistent patterns observed across other datasets.



(a) The relative error curve during warm-up.



(b) The relative error curve during inference.

Fig. 7: Comparison of IGNN-Solver with **GN**, **NN**, **GRU**, and **TCN**. Our proposed IGNN-Solver (with **GN**) achieves the fastest convergence and superior overall performance, owing to its lightweight design and ability to effectively utilize rich graph information.

standard iterator step. This explains the observed improvement in inference efficiency of at least 1.5× in Section V-B.

E. Cost and amortization analysis

To quantify the computational overhead introduced by the learnable solver, Figure 6 illustrates the proportion of training time of IGNN-Solver relative to that of IGNN. Owing to the compact design of the solver module, the additional training overhead remains extremely small, accounting for only about 1% of the overall training time.

Furthermore, as summarized in Table V, we conduct an amortization analysis by comparing IGNN-Solver with standard Picard iteration and AA method. While the joint optimization of the solver and predictor introduces a marginal training overhead (approximately 0.05× per epoch), IGNN-Solver delivers a pronounced inference-time advantage, achieving a 3.7× to 6.3× speedup per epoch across all large-scale datasets. In particular, on the `ogbn-papers100M` dataset, our approach reduces the inference time from 227.69s to 36.24s while the peak GPU memory usage of our model is comparable to that of recent implicit baselines.

F. Other choices for graph neural layers in IGNN-Solver

We provide additional evidence on the convergence and generalization of IGNN-Solver by exploring alternatives to our proposed graph neural (GN) layers, including neural network (NN), temporal convolutional network (TCN) [74], and gated recurrent unit (GRU) [75]. The relative residual curves on the Citeseer dataset, observed during both the warm-up and final inference stages, are presented in Figure 7. Interestingly, despite their larger number of parameters and more complex architectures, TCN and GRU consistently underperform compared to NN. Notably, IGNN-Solver (GN) significantly improves the convergence path of all solvers and exhibits the fastest rate of residual reduction, establishing itself as the most effective predictor in IGNN-Solver due to its ability to effectively exploit rich graph information.

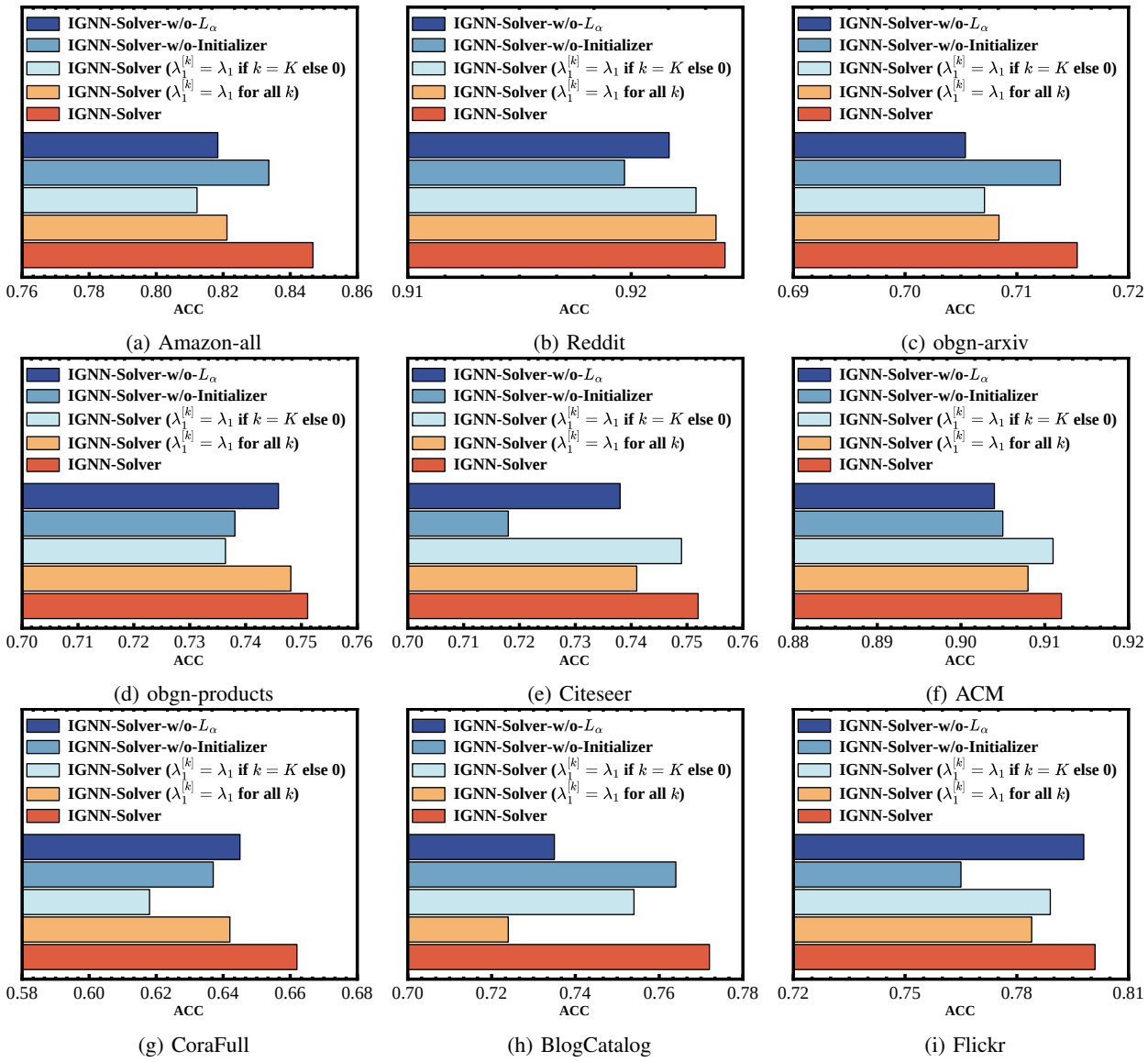


Fig. 8: Ablation studies for IGNN-Solver. This study shows that all components contribute significantly to accelerating the convergence process.

G. Ablation study

In this section, we analyze the effect of different loss components on the IGNN-Solver. The performance of IGNN-Solver and its variants on the Citeseer dataset is shown in Figure 8, with similar patterns observed across other datasets and solvers. For the main convergence loss $\mathcal{L}_{\text{conv}}$ in fixed-point iterations (18), we design two contrasting schemes concerning its weight $\lambda_1^{[k]}$s: setting them to a constant value (i.e., $\lambda_1^{[k]} = \lambda_1$ for all k), or setting all values except the K-th term to zero (i.e., $\lambda_1^{[k]} = \lambda_1$ if $k = K$ else 0). Both variant solvers still perform well, yet our suggested monotonically increasing scheme (i.e., emphasizing the later steps to a greater extent) exhibits the best performance.

Furthermore, removing either the initializer or the alpha loss $\mathcal{L}_{\boldsymbol{\alpha}}$ negatively impacts performance, with the former having a more detrimental impact on the solver. Nonetheless, all these ablation configurations significantly outperform methods without a solver, underscoring the advantages of employing a custom learnable solver for the IGNN model.

VI. Conclusion

This paper introduces IGNN-Solver, a novel graph neural solver tailored for achieving fast fixed-point solving in IGNNs. Unlike previous approaches that impose constraints on the structure or parameters of the implicit layer, we propose to leverage the important idea that implicit models separate the representation from the forward computation. By leveraging the generalized Anderson Acceleration method and parameterizing it with a tiny GNN, IGNN-Solver learns iterative updates as a graph-dependent temporal process. To address large-scale graph tasks, we further tailored sparsification and storage compression methods for the IGNN-Solver and integrated them into its design. Our extensive experiments demonstrate the significant acceleration in inference achieved by IGNN-

Solvers, with a speedup ranging from $1.5\times$ to $8\times$, while maintaining accuracy. Notably, this acceleration is particularly pronounced as the scale of the graph increases. These findings underscore the potential of IGNN-Solver for large-scale end-to-end deployment in real-world applications.

Looking ahead, IGNN-Solver opens new avenues for the integration of learnable solvers in implicit GNNs, bridging the gap between solver efficiency and adaptability to diverse graph structures. Additionally, investigating the interplay between solver generalization and graph topology holds promise for uncovering deeper insights into the dynamics of graph-based learning. With its ability to combine speed, scalability, and accuracy, IGNN-Solver represents a pivotal step toward making implicit graph models a viable choice for large-scale real-world systems. We hope that IGNN-Solver inspires further innovations in learnable solvers for IGNNs, driving progress in scalable and efficient algorithms on graphs.

REFERENCES

[1] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui, "Implicit graph neural networks," Advances in Neural Information Processing Systems, vol. 33, pp. 11 984–11 995, 2020.

[2] J. Liu, K. Kawaguchi, B. Hooi, Y. Wang, and X. Xiao, "Eignn: Efficient infinite-depth graph neural networks," Advances in Neural Information Processing Systems, vol. 34, pp. 18 762–18 773, 2021.

[3] J. Baker, Q. Wang, C. D. Hauck, and B. Wang, "Implicit graph neural networks: a monotone operator viewpoint," in International Conference on Machine Learning. PMLR, 2023, pp. 1521–1548.

[4] Q. Chen, Y. Wang, Y. Wang, J. Chang, Q. Tian, J. Yang, and Z. Lin, "Efficient and scalable implicit graph neural networks with virtual equilibrium," in 2022 IEEE International Conference on Big Data (Big Data). IEEE, 2022, pp. 864–873.

[5] Q. Li, Z. Han, and X. M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in 32nd AAAI Conference on Artificial Intelligence, AAAI 2018. AAAI press, 2018, pp. 3538–3545.

[6] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in International Conference on Learning Representations, 2020.

[7] X. Zhang, Y. Xu, W. He, W. Guo, and L. Cui, "A comprehensive review of the oversmoothing in graph neural networks," in CCF Conference on Computer Supported Cooperative Work and Social Computing. Springer, 2023, pp. 451–465.

[8] J. Liu, B. Hooi, K. Kawaguchi, Y. Wang, C. Dong, and X. Xiao, "Scalable and effective implicit graph neural networks on large graphs," in The Twelfth International Conference on Learning Representations, 2024. [Online]. Available: https://openreview.net/forum?id=QcMdPYBwTu

[9] Z. Geng, X.-Y. Zhang, S. Bai, Y. Wang, and Z. Lin, "On training implicit models," Advances in Neural Information Processing Systems, vol. 34, pp. 24 247–24 260, 2021.

[10] Y. Yang, T. Liu, Y. Wang, Z. Huang, and D. Wipf, "Implicit vs unfolded graph neural networks," arXiv preprint arXiv:2111.06592, 2021.

[11] M. Nastorg, M.-A. Bucci, T. Faney, J.-M. Gratien, G. Charpiat, and M. Schoenauer, "An implicit gnn solver for poisson-like problems," arXiv preprint arXiv:2302.10891, 2023.

[12] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in 5th International Conference on Learning Representations (ICLR). OpenReview.net, 2017.

[13] J. Liu, B. Hooi, K. Kawaguchi, and X. Xiao, "Mgnni: Multiscale graph neural networks with implicit layers," Advances in Neural Information Processing Systems, vol. 35, pp. 21 358–21 370, 2022.

[14] J. Liu, B. Hooi, K. Kawaguchi, Y. Wang, C. Dong, and X. Xiao, "Scalable and effective implicit graph neural networks on large graphs," in The Twelfth International Conference on Learning Representations, 2024.

[15] S. Bai, V. Koltun, and J. Z. Kolter, "Neural deep equilibrium solvers," in International Conference on Learning Representations, 2021.

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

[17] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," Advances in neural information processing systems, vol. 32, 2019.

[18] S. Bai, V. Koltun, and J. Z. Kolter, "Multiscale deep equilibrium models," in Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020, pp. 5238–5250.

[19] S. Yu, F. Alesiani, W. Yin, R. Jenssen, and J. C. Principe, "Principle of relevant information for graph sparsification," in Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, ser. Proceedings of Machine Learning Research, J. Cussens and K. Zhang, Eds., vol. 180. PMLR, 01–05 Aug 2022, pp. 2331–2341. [Online]. Available: https://proceedings.mlr.press/v180/yu22c.html

[20] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in International Conference on Machine Learning. PMLR, 2020, pp. 11 458–11 468.

[21] M. Köppen, "The curse of dimensionality," in 5th online world conference on soft computing in industrial applications (WSC5), vol. 1, 2000, pp. 4–8.

[22] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics, 2012, pp. 1–8.

[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," Advances in neural information processing systems, vol. 30, 2017.

[24] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," Advances

in neural information processing systems, vol. 33, pp. 22 118–22 133, 2020.

[25] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," in Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc., 2019, pp. 11 960–11 970. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf

[26] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in International Conference on Learning Representations, 2018, pp. 1–13.

[27] Z. Meng, S. Liang, H. Bao, and X. Zhang, "Co-embedding attributed networks," in Proceedings of the twelfth ACM international conference on web search and data mining, 2019, pp. 393–401.

[28] J. Wu, J. He, and J. Xu, "Demo-net: Degree-specific graph neural networks for node and graph classification," in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 406–415. [Online]. Available: https://doi.org/10.1145/3292500.3330950

[29] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," Advances in neural information processing systems, vol. 33, pp. 22 092–22 103, 2020.

[30] K. Wang, Z. Shen, C. Huang, C.-H. Wu, Y. Dong, and A. Kanakia, "Microsoft academic graph: When experts are not enough," Quantitative Science Studies, vol. 1, no. 1, pp. 396–413, 2020.

[31] C. Park, D. Kim, J. Han, and H. Yu, "Unsupervised attributed multiplex network embedding," in Proceedings of the AAAI conference on artificial intelligence, vol. 34, no. 04, 2020, pp. 5371–5378.

[32] Y. Zhong, H. Vu, T. Yang, and B. Adhikari, "Efficient and effective implicit dynamic graph neural network," in Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2024, pp. 4595–4606.

[33] A. Choudhuri, Y. Zhong, and B. Adhikari, "Implicit hypergraph neural network," in 2025 IEEE International Conference on Big Data (BigData), 2025, pp. 510–519.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," International Conference on Learning Representations, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[35] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017, pp. 5115–5124.

[36] X. Wang, M. Zhu, D. Bo, P. Cui, C. Shi, and J. Pei, "Am-gcn: Adaptive multi-channel graph convolutional networks," in Proceedings of the 26th ACM SIGKDD International conference on knowledge discovery & data mining, 2020, pp. 1243–1253.

[37] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," Advances in neural information processing systems, vol. 31, 2018.

[38] P. H. Avelar, A. R. Tavares, M. Gori, and L. C. Lamb, "Discrete and continuous deep residual learning over graphs," arXiv preprint arXiv:1911.09554, 2019.

[39] M. Poli, S. Massaroli, J. Park, A. Yamashita, H. Asama, and J. Park, "Graph neural ordinary differential equations," arXiv preprint arXiv:1911.07532, 2019.

[40] L.-P. Xhonneux, M. Qu, and J. Tang, "Continuous graph neural networks," in International conference on machine learning. PMLR, 2020, pp. 10 432–10 441.

[41] B. Chamberlain, J. Rowbottom, M. I. Gorinova, M. Bronstein, S. Webb, and E. Rossi, "Grand: Graph neural diffusion," in International conference on machine learning. PMLR, 2021, pp. 1407–1418.

[42] M. Thorpe, T. Nguyen, H. Xia, T. Strohmer, A. Bertozzi, S. Osher, and B. Wang, "Grand++: Graph neural diffusion with a source term," ICLR, 2022.

[43] Z. Ling, X. Xie, Q. Wang, Z. Zhang, and Z. Lin, "Global convergence of over-parameterized deep equilibrium models," in International Conference on Artificial Intelligence and Statistics. PMLR, 2023, pp. 767–787.

[44] E. Winston and J. Z. Kolter, "Monotone operator equilibrium networks," Advances in neural information processing systems, vol. 33, pp. 10 718–10 728, 2020.

[45] Z. Ling, L. Li, Z. Feng, Y. Zhang, F. Zhou, R. C. Qiu, and Z. Liao, "Deep equilibrium models are almost equivalent to not-so-deep explicit models for high-dimensional Gaussian mixtures," in Proceedings of the 41st International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 21–27 Jul 2024, pp. 30 585–30 609. [Online]. Available: https://proceedings.mlr.press/v235/ling24a.html

[46] Q. Zhao, W. Ren, T. Li, X. Xu, and H. Liu, "Graphgpt: Graph learning with generative pre-trained transformers," arXiv preprint arXiv:2401.00529, 2023.

[47] Z. Huang, S. Bai, and J. Z. Kolter, "$(implicit)^2$: Implicit layers for implicit representations," Advances in Neural Information Processing Systems, vol. 34, pp. 9639–9650, 2021.

[48] 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. PMLR, 2017, pp. 1263–1272.

[49] E. K. Ryu and S. Boyd, "Primer on monotone operator methods," Appl. comput. math, vol. 15, no. 1, pp. 3–43, 2016.

[50] L. Jing, Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić, "Tunable efficient unitary neural networks (eunn) and their application to rnns," in International Conference on Machine Learning. PMLR, 2017, pp. 1733–1741.

[51] P. R. Halmos, "A hilbert space problem book," Graduate Texts in Mathematics, 1982.

[52] S. G. Krantz and H. R. Parks, The implicit function

theorem: history, theory, and applications. Springer Science & Business Media, 2002.

[53] B. Hui, D. Yan, X. Ma, and W.-S. Ku, "Rethinking graph lottery tickets: Graph sparsity matters," arXiv preprint arXiv:2305.02190, 2023.

[54] J. F. Lutzeyer, C. Wu, and M. Vazirgiannis, "Sparsifying the update step in graph neural networks," in Topological, Algebraic and Geometric Learning Workshops 2022. PMLR, 2022, pp. 258–268.

[55] S. Zhang, M. Wang, P.-Y. Chen, S. Liu, S. Lu, and M. Liu, "Joint edge-model sparse learning is provably efficient for graph neural networks," arXiv preprint arXiv:2302.02922, 2023.

[56] J. Zhao, X. Wang, C. Shi, B. Hu, G. Song, and Y. Ye, "Heterogeneous graph structure learning for graph neural networks," in Proceedings of the AAAI conference on artificial intelligence, vol. 35, 2021, pp. 4697–4705.

[57] M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park, "Hypersolvers: Toward fast continuous-depth models," Advances in Neural Information Processing Systems, vol. 33, pp. 21 105–21 117, 2020.

[58] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in International conference on machine learning. PMLR, 2019, pp. 6861–6871.

[59] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in International Conference on Learning Representations, 2018.

[60] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in International conference on machine learning. PMLR, 2018, pp. 5453–5462.

[61] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in International conference on machine learning. PMLR, 2020, pp. 1725–1735.

[62] S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup, "Is heterophily a real nightmare for graph neural networks to do node classification?" arXiv preprint arXiv:2109.05641, 2021.

[63] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[64] J. Zhao, M. Qu, C. Li, H. Yan, Q. Liu, R. Li, X. Xie, and J. Tang, "Learning on large-scale text-attributed graphs via variational inference," arXiv preprint arXiv:2210.14709, 2022.

[65] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in Artificial intelligence and statistics. PMLR, 2009, pp. 488–495.

[66] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," Advances in neural information processing systems, vol. 29, 2016.

[67] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in Proceedings of the AAAI conference on artificial intelligence, vol. 32, no. 1, 2018.

[68] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" arXiv preprint arXiv:1810.00826, 2018.

[69] C. Gallicchio and A. Micheli, "Fast and deep graph neural networks," in Proceedings of the AAAI conference on artificial intelligence, vol. 34, no. 04, 2020, pp. 3898–3905.

[70] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," Machine learning, vol. 102, pp. 209–245, 2016.

[71] J. Park, J. Choo, and J. Park, "Convergent graph solvers," arXiv preprint arXiv:2106.01680, 2021.

[72] Q. Chen, Y. Wang, Y. Wang, J. Yang, and Z. Lin, "Optimization-induced graph implicit nonlinear diffusion," in International Conference on Machine Learning. PMLR, 2022, pp. 3648–3661.

[73] D. P. Kingma, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[74] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 156–165.

[75] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, D. Wu, M. Carpuat, X. Carreras, and E. M. Vecchi, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. [Online]. Available: https://aclanthology.org/W14-4012

[76] S. Oltra and O. Valero, "Banach's fixed point theorem for partial metric spaces," Rend. Istit. Mat. Univ. Trieste, vol. 36, pp. 17–26, 2004.

[77] H. L. Royden and P. Fitzpatrick, Real analysis. Macmillan New York, 1988, vol. 32.

[78] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, 2015, pp. 1365–1374.

[79] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," Annals of operations research, vol. 134, pp. 19–67, 2005.

[80] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in Proceedings of the AAAI conference on artificial intelligence, vol. 34, no. 04, 2020, pp. 5363–5370.

[81] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in International conference on neural information processing. Springer, 2018, pp.

362–373.
[82] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in Proceedings of the 13th international conference on web search and data mining, 2020, pp. 519–527.
[83] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in International conference on neural information processing. Springer, 2018, pp. 362–373.
[84] J. Zhou and Q. Yu, "Dcrnn: A deep cross approach based on rnn for partial parameter sharing in multi-task learning," arXiv preprint arXiv:2310.11777, 2023.
[85] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," arXiv preprint arXiv:2002.07962, 2020.
[86] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," arXiv preprint arXiv:2006.10637, 2020.
[87] X. Zhao, F. Chen, and J.-H. Cho, "Deep learning for predicting dynamic uncertain opinions in network data," in 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018, pp. 1150–1155.
[88] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," arXiv preprint arXiv:1809.10341, 2018.
[89] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.
[90] Y. Ma, S. Wang, C. C. Aggarwal, D. Yin, and J. Tang, "Multi-dimensional graph convolutional networks," in Proceedings of the 2019 siam international conference on data mining. SIAM, 2019, pp. 657–665.
[91] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in The world wide web conference, 2019, pp. 2022–2032.
[92] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," Advances in neural information processing systems, vol. 30, 2017.

Junchao Lin received the B.S. and M.S. degrees in mathematics from Wuhan University of Technology in 2020 and 2023, respectively. He is currently pursuing the Ph.D. degree with the School of EIC, Huazhong University of Science and Technology. His research interests include graph neural networks, artificial intelligence, machine learning, and deep learning.

Zenan Ling (Member, IEEE) is currently an assistant professor at Huazhong University of Science and Technology, School of Electronic Information and Communications. Prior to this, he was a postdoctoral researcher at Peking University, School of Intelligence Science and Technology, under the supervision of Prof. Zhouchen Lin. He received his Ph.D. in 2020 from Shanghai Jiao Tong University, where he worked under the supervision of Prof. Robert C. Qiu. He earned his B.S. degree in Mathematics from Nanjing University in 2015. His research interests broadly include machine learning, signal processing, random matrix theory, and high-dimensional statistics. His research has been funded by the National Natural Science Foundation of China and the Natural Science Foundation of Hubei Province.

Zhanbo Feng (Student Member, IEEE) received the B.S. degree in computer science and technology from Dalian University of Technology, China, in 2020. He is currently working toward the Ph.D. degree with Shanghai Jiao Tong University. His current research interests include machine learning, reinforcement learning, wireless communication and image generation.

Jingwen Xu received the B.S. degree in mathematics from Wuhan University of Technology, China, in 2020. She is currently pursuing the Ph.D. degree in physics at Wuhan University of Technology. Her current research interests include machine learning, differential equations, and numerical computation.

Minxuan Liao received the B.S. degree in information and communication engineering from Huazhong University of Science and Technology, China, in 2020. He is currently working toward the MS degree with Huazhong University of Science and Technology. His current research interests include machine learning, generative models and implicit models.

Feng Zhou received his Ph.D. degree in computer science from School of Computer Science and Engineering, University of New South Wales, Sydney, Australia. He was a Postdoctoral researcher at Department of Computer Science and Technology, Tsinghua University, Beijing, China, and now is a lecturer at Center for Applied Statistics and School of Statistics, Renmin University of China, Beijing, China. His research interests include statistical machine learning, stochastic processes, Bayesian methods, and AI4science applications.

Tianqi Hou received the Ph.D. degree in physics from HKUST, HongKong, in 2021. He is now a researcher at Lagrange Mathematics and Computing Research Center of Huawei in Paris. His current research interests include machine learning and statistical physics.

Zhenyu Liao (Member, IEEE) received the B.E. degree in optoelectronics from the Huazhong University of Science and Technology (HUST), China, in 2014, the master's degree in signal and image processing, and the Ph.D. degree in mathematics and informatics from Université Pairs-Saclay, France, in 2016 and 2019, respectively. He then worked as postdoctoral fellow with the Department of Statistics and with the International Computer Science Institute (ICSI) of University of California, Berkeley, USA. He is now an associate professor with the School of Electronic Information and Communications, HUST, China. His research interests include machine learning theory and methods, as well as their interactions with high-dimensional statistics and random matrix theory. He coauthored the book "Random Matrix Methods for Machine Learning" published with Cambridge University Press. He served as an area chair of ICLR 2025.

Robert Caiming Qiu (Fellow, IEEE) received the Ph.D. degree in electrical engineering from New York University (former Polytechnic University) Brooklyn, NY, USA. He joined the School of Electronic Information and Communications, Huazhong University of Science and Technology (HUST), China, as a full professor, in 2020. Before joining HUST, he was an associate professor with the Department of Electrical and Computer Engineering, Center for Manufacturing Research, Tennessee Tech University, Cookeville, TN, USA, in 2003, where he became a professor in 2008. He founded and served as the CEO and President of Wiscom Technologies Inc., a company specializing in manufacturing and marketing WCDMA chipsets. He was with GTE Laboratories Inc. (now Verizon), Waltham, MA, USA, and Bell Laboratories, Lucent, Whippany, NJ, USA. He has coauthored Cognitive Radio Communication and Networking: Principles and Practice (JohnWiley, 2012) and Cognitive Networked Sensing: A Big Data Way (Springer, 2013) and authored BigData and Smart Grid (JohnWiley, 2015). Hehas authored more than 100 journal articles/book chapters and 120 conference papers. Furthermore, he has made 15 contributions to 3GPP and IEEE standards bodies. He has served as a TPC Member for GLOBECOM, ICC, WCNC, and MILCOM. He has also served as an associate editor for IEEE Transactions on Vehicular Technology and other international journals.