GRAPH LOW-RANK ADAPTERS OF HIGH REGULARITY FOR GRAPH NEURAL NETWORKS AND GRAPH TRANSFORMERS

Pantelis Papageorgiou[†]
Anthropocene Bio & University of Oxford

Haitz Sáez de Ocáriz Borde[†]
University of Oxford

Anastasis Kratsios[†]
McMaster University & Vector Institute

Michael Bronstein
University of Oxford & AITHYRA

ABSTRACT

We introduce a new low-rank graph adapter, GConv-Adapter, that leverages a two-fold normalized graph convolution and trainable low-rank weight matrices to achieve state-of-the-art (SOTA) and near-SOTA performance in GNN fine-tuning for standard message-passing neural networks (MPNNs) and graph transformers (GTs) in both inductive and transductive learning. We motivate our design by deriving an upper bound on the adapter's Lipschitz constant for δ-regular random (expander) graphs, and we compare it against previous methods which we show to be unbounded.

1 INTRODUCTION

Low-rank adapters (LoRAs) (Hu et al., 2022a) were originally introduced to fine-tune large language models (LLMs) under constrained computational budgets. While current graph neural network (GNN) models are smaller than leading LLMs, they are expected to grow in size as graph foundation models continue to evolve (Liu et al., 2023; Mao et al., 2024). Recently, LoRA-type adapters specialized for GNNs have been proposed, notably AdapterGNN (Li et al., 2024) and G-Adapter (Gui et al., 2024), which, like standard LoRAs, can be trained with a small number of parameters using low-rank matrix factorization. Although these techniques represent an important first step, they still have limitations. Specifically, we show that AdapterGNN is insensitive to input graph features, while G-Adapter can lead to unstable learners for certain extremal graphs. Motivated by these shortcomings, we introduce the *GConv-Adapter*. Specifically, our GConv-Adapter is capable of capturing second-hop interactions in the graph (Proposition 3) and produces stable learners with a bounded Lipschitz constant for graphs with sufficient connectivity (Theorem 1). We validate our framework across a large number of GNN benchmarks in Section 4.

2 PRELIMINARIES AND RELATED WORK

Graphs A graph can be defined as the tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E} \subseteq (\mathcal{V} \times \mathcal{V})$ is the set of edges in the graph. An edge between nodes $v_i, v_j \in \mathcal{V}$ is denoted by $(v_i, v_j) \in \mathcal{E}$. The one-hop neighborhood of a node v_i, denoted as $\mathcal{N}(v_i) = \{v_j \mid (v_i, v_j) \in \mathcal{E}\}$, includes all nodes directly connected to v_i and thus defines the local connectivity around the node. The connectivity of a graph with $N = |\mathcal{V}|$ nodes can be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where A_{ij} indicates the connection strength between nodes v_i and v_j.

Definition 1 (δ-Regular Graphs)**.** *A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ on N vertices (or nodes) is said to be* δ-regular *if every vertex has the same degree, namely* $\deg(v) = \delta, \quad \forall\, v \in \mathcal{V}$.

Email addresses: pantelis.papageorgiou@anthropocene.bio, chri6704@ox.ac.uk, kratsioa@mcmaster.ca, michael.bronstein@cs.ox.ac.uk

[†]Equal contribution

In this case, the minimal degree, maximal degree, and average degree are all equal to δ. Consequently, any bound or property of the graph that might otherwise depend on the degree becomes uniform across all vertices; this observation is relevant to Section 3 and Appendix A.

Definition 2 (Expander Graphs)**.** *A family of graphs $\{\mathcal{G}_N\}_{N\in\mathbb{N}}$, where each $\mathcal{G}_N = (V_N, E_N)$ has $|V_N| = N$, is called an* expander family *if there exists a constant $\varepsilon > 0$ independent of N such that for every $\mathcal{G}_N$ and every subset $S \subseteq V_N$ with $|S| \leq \frac{N}{2}$, the edge boundary ∂S satisfies $|\partial S| \geq \varepsilon\,|S|$. Here, ∂S is defined as the set of edges with one endpoint in S and the other in $V_N \setminus S$.*

An alternative spectral characterization involves the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix ($D_{i,i} = \deg(v_i)$). For a δ-regular graph, if λ_2 denotes the second smallest eigenvalue of $\mathbf{L}$ (the algebraic connectivity), then the graph is an expander if there exists a constant $c > 0$ such that $\lambda_2 \geq c$. Many constructions of expander graphs yield δ-regular (or nearly regular) graphs, in which case the Lipschitz constant bound for the GConv-Adapter becomes degree-independent (as we will see in Theorem 7). However, the notion of an expander is broader and can include irregular graphs that nevertheless exhibit strong connectivity.

Message Passing Neural Networks (MPNNs) and Graph Transformers (GTs). For graphs with node features, such as those considered in this work, each node v_i has an associated feature vector $\mathbf{x}_i \in \mathbb{R}^{1\times D}$, where D is the feature space dimension. By stacking these vectors along the row dimension, they form a feature matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$. Message Passing Neural Networks (MPNNs) are a foundational class of GNNs that operate by exchanging messages between node neighborhoods to iteratively refine their feature representations. Following (Bronstein et al., 2021), in layer l of an MPNN (excluding edge-level features for simplicity), the updated feature representation $\mathbf{x}_i^{(l+1)}$ for node v_i is: $\mathbf{x}_i^{(l+1)} = \phi\left(\mathbf{x}_i^{(l)}, \bigoplus_{j\in\mathcal{N}(v_i)} \psi\left(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}\right)\right)$. Here, ψ is a message function, $\bigoplus$ denotes a permutation-invariant aggregation function, such as summation, mean, or max, and ϕ is an update or readout function. Both ψ and ϕ are typically parameterized by neural networks, such as Multi-Layer Perceptrons (MLPs), and are learned during training. This update process is local to each node's one-hop neighborhood, meaning that each node communicates only with its immediate neighbors at each layer. On the other hand, GTs are a class of models that adapt the Transformer architecture (Vaswani et al., 2017), originally designed for natural language processing (NLP), to the domain of graph representation learning. GTs treat nodes as analogous to tokens in a sequence, with the self-attention mechanism facilitating the exchange of information across the entire graph. This allows every node to attend to every other node within each layer, effectively treating the graph as fully connected (Bronstein et al., 2021). While this facilitates capturing long-range dependencies, it discards the locality inductive bias inherent in graph data, which can be crucial in low-data regimens. Additionally, one of the significant challenges with GTs is their computational complexity, particularly for large-scale graphs, where the self-attention mechanism has a quadratic complexity in the number of nodes $\mathcal{O}(N^2)$ (Ying et al., 2021; Rampášek et al., 2022; Borde et al., 2024).

Pre-training and Fine-tuning Graph Neural Networks. Let a GNN encoder, either an MPNN or a GT, denoted as f_Θ, encode a graph $\mathcal{G}$ into a feature embedding $\mathbf{H} = f_\Theta(\mathbf{X}, \mathbf{A}, \mathbf{E}) \in \mathbb{R}^{N\times D_H}$, where Θ represents the model parameters. Pre-training aims to initialize Θ_{pre} by minimizing a loss function $\mathcal{L}_S$ over a large, generic dataset $\mathcal{D}_S$, with the objective: $\Theta_{\text{pre}} = \underset{\Theta}{\operatorname{argmin}} \sum_{(\mathcal{G}_i^S, \mathcal{Y}_i^S)\in\mathcal{D}_S} \mathcal{L}_S\left(g_\Psi \circ f_\Theta(\mathcal{G}_i^S), \mathcal{Y}_i\right)$, where, for instance, $\mathcal{Y}_i$ represents the labels associated with graph $\mathcal{G}_i^S \in \mathcal{D}_S$. Specifically, $\mathcal{Y}_i^S = \mathbf{Y}_i \in \mathbb{R}^{N\times 1}$ is a set of node labels in the case of node-level classification, while $\mathcal{Y}_i^S = y_i \in \mathbb{N}$ denotes the label for the entire graph in graph-level classification. Other tasks such as link prediction, and regression are also possible. The function g_Ψ is a task-specific classifier parameterized by Ψ which is composed with the GNN encoder f_Θ to yield the final output. Note that in practice Θ and Ψ are optimized concurrently. Full fine-tuning begins with the model initialized with the pre-trained weights Θ_{pre} and updates these weights to $\Theta_{\text{pre}} + \Delta\Theta$ using $\mathcal{L}_T$ over a smaller, task-specific dataset $\mathcal{D}_T$: $\Delta\Theta^* = \underset{\Delta\Theta}{\operatorname{argmin}} \sum_{(\mathcal{G}_j^T, \mathcal{Y}_j^T)\in\mathcal{D}_T} \mathcal{L}_T\left(g'_\Psi \circ f_{\Theta_{\text{pre}}+\Delta\Theta}(\mathcal{G}_j^T), \mathcal{Y}_j^T\right)$, where in general, the loss $\mathcal{L}_T$ may be different from $\mathcal{L}_S$, the labels in $\mathcal{D}_T$ could be at a different level (graph, node, edge) as compared to the original dataset $\mathcal{D}_S$, and g'_Ψ can also be a different classifier. One of the main drawbacks of full fine-tuning is that for each downstream task, a different set of parameters $\Delta\Theta$ is learned, where the total parameter count $|\Delta\Theta|$ equals that of the original model $|\Theta_{\text{pre}}|$. This means that if

the pre-trained GNN model is large, retraining, storing, and deploying multiple fine-tuned models can become challenging, particularly in resource-constrained environments. Although current GNN models are not as large as the leading LLMs, there is an expectation that they will continue to increase in size with the development of graph foundation models (Liu et al., 2023; Mao et al., 2024). Consequently, this work establishes a foundation for addressing these forthcoming challenges, ensuring that the methods developed today will remain effective as GNN models become larger.

Related Work Approaches that reduce trainable parameters during fine-tuning while preserving performance are known as Parameter-Efficient Fine-Tuning (PEFT). (He et al., 2022; Pfeiffer et al., 2020; Ding et al., 2022; Yu et al., 2022; Han et al., 2024). Such approaches assume a relatively mild distribution shift between the original dataset $\mathcal{L}_S$ and the fine-tuning dataset $\mathcal{L}_T$, which is generally the case in the context of language. Based on this assumption, PEFT methods conjecture that retraining only a small subset of parameters will be sufficient to adapt the model to new tasks. A prominent example within PEFTs is the Adapter framework (Houlsby et al., 2019), which operates by injecting a set of new trainable layers either sequentially or in parallel to a Transformer's Multi-Head Attention (MHA) or MLPs. Fine-tuning only these new layers, the adapters, results in relatively inexpensive retraining of the overall network. AdapterDrop (Rücklé et al., 2021) prunes adapter blocks in lower layers. AdapterFusion (Pfeiffer et al., 2021) enhances multi-task learning by facilitating knowledge sharing across tasks through a two-stage process. Compacter and Compacter++ (Mahabadi et al., 2021) further reduce the proportion of trainable parameters by leveraging techniques like the Kronecker product and weight-sharing. Additional works in this line of research include those by Wang et al. (2021); Karimi Mahabadi et al. (2021); Fu et al. (2022); Wang et al. (2022). On the other hand, LoRA (Hu et al., 2022a; Zhu et al., 2024) leverages the insight that updates required during fine-tuning often have a low intrinsic rank (Aghajanyan et al., 2021). Rather than adding new trainable layers, LoRA fine-tunes only a small number of parameters by introducing two low-rank matrices that approximate the changes in the query and value weight matrices within the Transformer's MHA mechanism. Similarly, BitFit (Ben Zaken et al., 2022) reduces the parameter footprint even further by updating only the bias terms of the model, offering a highly parameter-efficient solution. Also, combining multiple PEFT techniques has been a growing area of research: (Hu et al., 2022b; Mao et al., 2022; Chen et al., 2023; Jiang et al., 2023) explore hybrid mechanisms, and He et al. (2022) offers a comprehensive review. While PEFT has been extensively explored in NLP, its application to GNNs is still in its early stages. Two recent efforts are AdapterGNN (Li et al., 2024), for MPNNs, and G-Adapter (Gui et al., 2024), which targets GTs.

3 METHODOLOGY

An effective GNN fine-tuning framework must address key desiderata: leveraging geometric priors to utilize new graph structures and avoid data inefficiency; maintaining the lightweight nature of PEFTs; ensuring integrability across GNN variants such as MPNNs, and GTs.

3.1 ON THE THEORETICAL LIMITATIONS OF ADAPTERGNN AND G-ADAPTER

For proofs regarding the theoretical considerations discussed in this section refer to Appendix A.

AdapterGNN. AdapterGNN injects an adapter layer with learnable parameters parallel to the GNN's MLP layers (please refer to Figure 3 in Li et al. (2024)):

$$A_{\text{AdapterGNN}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) = \text{BN}\left(\sigma\left(\mathbf{X}\mathbf{W}_{down}\right)\mathbf{W}_{up}\right), \tag{1}$$

where $\mathbf{X} \in \mathbb{R}^{N\times D}$ is the node feature matrix, $\mathbf{W}_{down} \in \mathbb{R}^{D\times r}$ and $\mathbf{W}_{up} \in \mathbb{R}^{r\times D}$ are the down-projection and up-projection matrices with $r \ll D$, $\sigma(\cdot)$ is a nonlinear activation function (e.g., ReLU), and BN denotes batch normalization. Notably, AdapterGNN does not utilize the adjacency matrix $\mathbf{A}$, thus ignoring the graph's structural information during fine-tuning. This lack of structural integration may hinder performance on tasks where the graph topology is crucial. Next, this inability to detect and process a graph's structure is formalized by the following proposition.

Proposition 1 (AdapterGNN is graph-agnostic)**.** *Let $r, N, D \in \mathbb{N}_+$, with $r \leq D$, σ be any activation function, and fix (low) rank r-matrices $\mathbf{W}_{down} \in \mathbb{R}^{D\times r}$ and $\mathbf{W}_{up} \in \mathbb{R}^{r\times D}$. For any graphs $\mathcal{G}, \mathcal{G}'$ on N vertices the following graph agnosticism property holds*

$$A_{\textit{AdapterGNN}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) = A_{\textit{AdapterGNN}}(\mathbf{X}|\mathcal{G}', \mathbf{W}_{down}, \mathbf{W}_{up}) \quad (\forall \mathbf{X} \in \mathbb{R}^{N\times D}). \tag{2}$$

G-Adapter. G-Adapter incorporates the graph's structure by applying a single graph convolution using the adjacency matrix within the adapter. The transformation can be written as:

$$A_{\text{G-Adapter}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}) \overset{\text{def.}}{=} \text{LN}\left(\mathbf{X}' + \sigma\left(\tilde{\mathbf{A}}\,\mathbf{X}'\,\mathbf{W}_{down}\,\mathbf{W}_{up}\right)\right), \quad \mathbf{X}' = \text{LN}\left(\mathbf{X}\right), \quad (3)$$

where $\tilde{\mathbf{A}} = \mathbf{A} + I_N$, with $\mathbf{A} = \mathbf{A}(\mathcal{G}) \in \mathbb{R}^{N\times N}$ which is the (unnormalized) adjacency matrix derived from the graph $\mathcal{G}$, $\mathbf{W}_{down}$ and $\mathbf{W}_{up}$ are as before, $\sigma(\cdot)$ is a nonlinear activation function, and LN denotes layer normalization. While G-Adapter explicitly introduces graph structure, it does not use the normalized adjacency matrix, which can lead to numerical instabilities and overemphasize nodes with high degrees. Moreover, the single application of the adjacency matrix limits the adapter's ability to capture higher-order neighbourhood information or complex dependencies in the graph. This static and shallow integration may restrict the model's flexibility in leveraging task-specific structural information. In short, although the G-Adapter incorporates the graph's topology into its predictions, the way in which it does so fails to produce adapters which are stable with respect to their input feature matrices. This is shown by the following lower bound on the Lipschitz regularity (linear stability with respect to its inputs) for certain random expander graph inputs.

Proposition 2 (The G-Adapter can have a diverging Lipschitz constant)**.** *Let $\sigma = \text{ReLU}$, $N, \delta \in \mathbb{N}_+$ with $\delta \leq N-1$, let $\mathcal{G}$ be any δ-regular random (expander) graph on N-vertices, and replace* LN *by the identity in equation 3, i.e., $\mathbf{X} \mapsto \mathbf{X} + \text{ReLU}\left(\tilde{\mathbf{A}}\,\mathbf{X}\,\mathbf{W}_{down}\,\mathbf{W}_{up}\right)$. Then, there are $D, r \in \mathbb{N}_+$ with $r \leq N$, $\mathbb{N}_+$, rank r matrices $\mathbf{W}_{down} \in \mathbb{R}^{D\times r}$ and $\mathbf{W}_{up} \in \mathbb{R}^{r\times D}$ such that*

$$\text{Lip}\left(A_{\text{G-Adapter}}(\cdot|\mathcal{G}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}})\right) \geq (\delta + 2)\ \textit{a.s.} \quad (4)$$

In particular, if $N - 1 = \delta \to \infty$ then, $\text{Lip}\left(A_{\textit{G-Adapter}}(\cdot|G, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}})\right) \in \Theta(N)$.

3.2 GConv-Adapter

To address the limitations of previous approaches, we propose GConv-Adapter:

$$A_{\text{G-Conv}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) \overset{\text{def.}}{=} \mathbf{X} + \alpha \cdot \text{Normalization}\left(\hat{\mathbf{A}}\,\sigma\left(\hat{\mathbf{A}}\,\mathbf{X}\,\mathbf{W}_{down}\right)\mathbf{W}_{up}\right), \quad (5)$$

where $\mathbf{X} \in \mathbb{R}^{N\times D}$ is the input node feature matrix, $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\,\tilde{\mathbf{D}}^{-1/2}$ is the symmetrically normalized adjacency matrix (using the degree matrix) with self-loops (hence the tildes), and $\mathbf{W}_{down} \in \mathbb{R}^{D\times r}$, $\mathbf{W}_{up} \in \mathbb{R}^{r\times D}$ are the learnable projection matrices with $r \ll D$. The non-linear activation $\sigma(\cdot)$, Normalization (LayerNorm or BatchNorm), and learnable scaling factor α together modulate the impact of the adapter-modified features relative to the original parameters. Recall that AdapterGNN cannot detect graph structure; this is not the case for the GConv-Adapter.

Proposition 3 (GConv-Adapter is not graph-agnostic)**.** *Let $\sigma = \text{ReLU}$ and $\text{Normalization}(Z) = Z$ for all $Z \in \mathbb{R}^{N\times D}$. There exists $r, N, D \in \mathbb{N}_+$ with $r \leq D$, and rank r matrices $\mathbf{W}_{down} \in \mathbb{R}^{D\times r}$, $\mathbf{W}_{up} \in \mathbb{R}^{r\times D}$ and $\alpha > 0$ such that: there exist N-vertex graphs $\mathcal{G}, \mathcal{G}'$ and a feature matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$ N such that: $A_{\textit{G-Conv}}$ distinguishes $\mathcal{G}$ and $\mathcal{G}'$; i.e.*

$$A_{\text{G-Conv}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) \neq A_{\text{G-Conv}}(\mathbf{X}|\mathcal{G}', \mathbf{W}_{down}, \mathbf{W}_{up}). \quad (6)$$

We now show that it can achieve an improved Lipschitz regularity compared to the G-Adapter. Namely, its Lipschitz constant is bounded for all regular (expander) graphs *independently of the graph degree* while that of the G-Adapter *diverged* as the graph degree and its number of nodes increased. In particular, note that while the next result permits normalization layers, we can set normalization to the identity, as in Proposition equation 2, to ensure a fair, apples-to-apples comparison.

Theorem 1 (GConv-Adapter's Lipschitz constant is degree-independent for expanders)**.** *Let $L_\sigma, \alpha \geq 0, \delta, r, N, D \in \mathbb{N}_+$, with $r \leq D$, σ be a L_σ-Lipschitz activation function, and Normalization be a 1-Lipschitz normalization (identity). For any δ-regular random (expander) graph $\mathcal{G}$ on N-vertices and any (low) rank r-matrices $\mathbf{W}_{down} \in \mathbb{R}^{D\times r}$ and $\mathbf{W}_{up} \in \mathbb{R}^{r\times D}$ we have*

$$\text{Lip}\left(A_{\text{G-Conv}}(\cdot|\mathbf{G}, \mathbf{W}_{down}, \mathbf{W}_{up})\right) \leq 1 + \alpha L_\sigma \|\mathbf{W}_{down}\|_{op}\|\mathbf{W}_{up}\|_{op} \quad \textit{a.s.} \quad (7)$$

If, instead, $\mathcal{G}$ is an arbitrary connected random graph then the right-hand side of equation 7 is at-most $1 + \alpha L_\sigma \frac{(1+\rho(\mathbf{A}))^2\|\mathbf{W}_{down}\|_{op}\|\mathbf{W}_{up}\|_{op}}{(1+\deg_-(\mathcal{G}))^2}$ a.s., where $\rho(\mathbf{A})$ is the spectral radius of the adjacency matrix, and $deg_-(\mathcal{G})$ is the minimal degree of the graph.

Hence, unlike G-Adapter, our GConv-Adapter is degree-independent and remains bounded for "graphs that are connected enough". The use of the symmetrically normalized adjacency matrix, as in GCNs (Kipf & Welling, 2017), ensures that feature propagation is properly scaled and prevents numerical instabilities from repeated adjacency matrix multiplications. Importantly, GConv-Adapter can be seamlessly integrated into classical GNN architectures, such as MPNNs and GTs, making it a versatile solution for a range of models. Only the projection matrices and scaling factor are trainable, maintaining the low parameter count typical of PEFTs. Moreover, by applying graph convolution before and after the non-linearity, GConv-Adapter allows nodes to aggregate information from their two-hop neighborhoods, enabling the model to capture higher-order structural patterns. We analyze the computational complexity of GConv-Adapter assuming a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes, $E = |\mathcal{E}|$ edges, input feature dimension D, and low-rank dimension $r \ll D$.

4 EXPERIMENTS

In the following section, we empirically validate our framework. We provide fine-grained results in Table 1 and Table 2 in the main text, as well as summary results in Tables 5 and 6 in Appendix D, accompanied by a more extensive discussion on performance comparison. Additionally, we also specify implementation details in Appendix C, and perform additional ablations to verify the different components of our adapter in Appendix E. The base configuration of GConv-Adapter follows the definition given in equation 5, the position of the adapters is sequential (before and after the GNN layers of the original model), for normalization we use BatchNorm for MPNNs and LayerNorm for GTs, and α, the learnable scaling parameter, is initialized to 1 at the beginning of training.

Inductive Learning Results GConv-Adapter consistently improves regression performance over full fine-tuning, see Table 1. For example, on ESOL it achieves an RMSE of 1.341 (a 3.73% improvement over full fine-tuning's 1.393), on FreeSolv an RMSE of 2.058 (14.49% improvement over 2.407), and on Lipophilicity an RMSE of 0.833 (4.90% improvement over 0.876). Overall, the average RMSE is reduced from 1.559 to 1.411 (a 9.49% improvement). Compared to other graph-specific PEFT methods, GConv-Adapter outperforms G-Adapter and is competitive with AdapterGNN. Similarly, GConv-Adapter also shows consistent improvements in classification. It achieves a ROC-AUC of 84.288% on Tox21 (1.80% higher than full fine-tuning's 82.795), 82.641% on SIDER (0.19% higher), and 97.558% on ClinTox (0.15% higher). Notably, on BACE, MUV, and HIV, it improves by 8.36%, 1.74%, and 3.13% respectively, leading to an overall average ROC-AUC of 81.142% versus 79.352% for full fine-tuning. Furthermore, GConv-Adapter outperforms other graph-specific PEFT methods—surpassing G-Adapter by 3.05% and AdapterGNN by 0.81%—and achieves substantial gains over classical PEFT techniques.

Transductive Learning Results Table 2 compares fine-tuning methods for node classification on the Cora, Citeseer, and PubMed networks using pre-trained NodeFormer (Wu et al., 2022) and DIFFormer-s (Wu et al., 2023) GT models pre-trained on ogbn-Arxiv (Hu et al., 2020a) (a larger citation network). For NodeFormer, our GConv-Adapter achieves the best accuracy on PubMed (89.162%) and competitive results on Cora and Citeseer, outperforming full fine-tuning and most PEFT baselines. On DIFFormer-s, GConv-Adapter delivers robust performance,achieving accuracies of 82.841% on Cora, 78.967% on Citeseer, and 88.615% on PubMed, substantially surpassing other graph-specific methods. Overall, these results demonstrate that GConv-Adapter effectively adapts pre-trained GTN models for node classification.

Table 1: Inductive Learning: Molecular Prediction Tasks. Results are reported as mean ± standard deviation of evaluation metrics (RMSE for regression tasks and ROC-AUC for classification tasks). Best, second-best, and third-best in red, blue, and brown.

Tuning Method	Pre-trained Model	Regression (RMSE ↓)				Classification (ROC-AUC ↑)						
		ESOL	FreeSolv	Lipophilicity	Avg.	Tox21	SIDER	ClinTox	BACE	MUV	HIV	Avg.
Full Fine-tuning	GCN	1.256±0.012	1.522±0.266	0.808±0.006	1.195	84.905±0.045	82.613±0.124	98.090±0.088	73.118±3.883	72.634±2.503	73.010±0.204	80.728
	GraphSAGE	1.267±0.041	2.169±0.193	0.824±0.007	1.420	85.346±0.120	83.357±0.301	97.270±0.089	62.586±3.758	75.054±0.766	73.862±0.525	79.579
	GAT	1.897±0.012	3.323±0.052	1.135±0.011	2.118	75.159±0.524	80.782±0.144	97.333±0.104	60.894±0.242	67.750±1.334	67.722±0.924	74.940
	GIN	1.153±0.011	2.612±0.883	0.737±0.011	1.501	85.770±0.295	83.811±0.165	96.923±0.112	76.566±2.037	73.713±0.952	76.186±0.855	82.162
	Avg.	1.393	2.407	0.876		82.795	82.481	97.404	68.291	72.288	72.695	
Surgical Fine-tuning	GCN	1.390±0.007	3.178±0.253	0.995±0.008	1.854	81.532±0.142	81.120±0.081	97.470±0.666	64.876±1.460	70.518±1.251	72.126±0.796	77.940
	GraphSAGE	1.521±0.073	2.885±0.016	0.987±0.019	1.798	82.660±0.072	82.157±0.257	97.568±0.065	57.915±4.329	70.418±2.510	69.428±1.291	76.691
	GAT	1.931±0.016	3.460±0.013	1.163±0.001	2.185	74.486±0.055	80.476±0.026	97.354±0.123	61.798±0.670	65.239±1.482	64.869±0.420	74.037
	GIN	1.454±0.007	2.967±0.066	1.028±0.001	1.816	81.456±0.111	81.640±0.207	97.093±0.213	60.575±3.048	70.799±1.667	66.913±1.475	76.413
	Avg.	1.574	3.123	1.043		80.034	81.348	97.371	61.291	69.244	68.334	
BitFit	GCN	2.334±0.001	5.122±0.182	1.205±0.002	2.887	67.288±0.417	79.081±0.056	96.646±0.336	50.803±8.271	48.057±8.621	59.732±3.433	66.934
	GraphSAGE	2.250±0.010	4.931±0.013	1.184±0.000	2.788	70.976±0.817	79.411±0.143	97.107±0.438	56.042±6.886	54.508±2.026	59.649±1.109	69.615
	GAT	2.333±0.016	5.405±0.296	1.204±0.000	2.981	67.956±0.713	79.043±0.013	96.721±0.103	62.198±2.310	56.732±1.654	49.217±3.051	68.645
	GIN	2.346±0.017	4.927±0.043	1.199±0.004	2.824	68.127±0.229	79.100±0.230	95.968±0.526	44.045±1.595	51.592±2.444	63.013±4.886	66.974
	Avg.	2.316	5.096	1.198		68.587	79.159	96.611	53.272	52.722	57.903	
LoRA	GCN	1.479±0.134	2.360±0.073	0.803±0.018	1.547	83.210±0.161	79.837±0.249	97.480±0.591	74.213±3.068	50.998±0.470	71.286±2.080	76.171
	GraphSAGE	1.280±0.032	2.255±0.369	0.731±0.006	1.422	85.034±0.555	83.295±0.373	96.416±0.173	67.240±3.416	62.411±1.783	72.624±1.113	77.837
	GAT	1.921±0.033	3.095±0.203	1.149±0.011	2.055	74.211±0.394	80.356±0.152	97.404±0.294	59.804±4.146	60.201±0.536	65.160±2.832	72.856
	GIN	1.248±0.013	2.700±1.075	0.719±0.011	1.556	85.760±0.444	83.438±0.443	96.833±0.346	78.241±1.238	66.208±1.499	74.017±1.771	80.749
	Avg.	1.482	2.603	0.851		82.054	81.731	97.033	69.874	59.954	70.772	
Adapter	GCN	1.228±0.023	1.459±0.064	0.778±0.022	1.155	85.123±0.282	81.884±0.312	97.179±0.413	73.483±2.375	70.782±0.507	72.511±1.703	80.160
	GraphSAGE	1.316±0.038	2.456±0.314	0.802±0.018	1.525	85.044±0.086	82.954±0.274	96.919±0.068	60.946±5.595	68.705±3.002	69.805±0.792	77.395
	GAT	1.906±0.033	3.419±0.062	1.118±0.030	2.148	75.237±0.313	80.369±0.092	97.483±0.096	62.812±2.439	64.208±1.669	62.096±3.325	73.701
	GIN	1.208±0.025	3.058±1.288	0.749±0.012	1.672	85.510±0.504	83.401±0.270	96.686±0.062	73.054±1.761	72.698±4.015	73.754±1.184	80.850
	Avg.	1.414	2.598	0.862		82.728	82.152	97.067	67.574	69.098	69.541	
G-Adapter	GCN	1.254±0.095	1.807±0.314	0.739±0.012	1.267	85.811±0.220	82.764±0.697	96.679±0.259	77.366±3.524	69.243±2.863	73.950±0.842	80.969
	GraphSAGE	1.240±0.046	1.939±0.444	0.735±0.021	1.305	86.107±0.156	83.253±0.995	96.977±0.434	77.314±1.329	66.958±3.040	74.646±0.806	80.876
	GAT	1.769±0.038	3.196±0.083	1.117±0.004	2.027	76.334±0.209	80.227±0.152	97.688±0.174	66.296±1.195	60.421±1.650	54.194±0.670	72.527
	GIN	1.233±0.055	2.550±0.279	0.731±0.003	1.505	85.119±0.846	83.611±0.229	97.131±0.510	75.239±1.903	68.768±1.236	73.623±0.855	80.582
	Avg.	1.374	2.373	0.835		83.343	82.464	97.119	**74.054**	66.347	69.103	
AdapterGNN	GCN	1.240±0.016	1.214±0.139	0.738±0.010	**1.064**	85.862±0.251	81.428±0.336	96.793±0.740	79.354±1.518	76.247±0.886	73.337±1.520	**82.170**
	GraphSAGE	1.239±0.024	1.552±0.030	0.729±0.011	**1.173**	84.980±0.126	82.981±0.143	96.964±0.389	72.045±2.303	75.736±1.045	75.326±1.320	81.339
	GAT	1.913±0.038	2.392±0.039	0.916±0.033	**1.740**	79.017±0.487	80.723±0.243	97.542±0.273	64.290±2.107	67.814±0.846	68.798±2.821	76.364
	GIN	1.229±0.023	1.594±0.174	0.737±0.001	1.187	86.048±0.448	82.826±0.225	97.243±0.196	76.508±1.358	75.397±0.065	74.467±1.478	82.082
	Avg.	1.405	**1.688**	**0.780**		83.977	81.989	97.136	73.049	**73.799**	72.982	80.489
GConv-Adapter (ours)	GCN	1.216±0.012	2.044±0.541	0.769±0.031	1.343	86.212±0.105	81.159±1.321	97.390±0.565	73.926±1.122	75.982±1.882	76.048±0.185	81.786
	GraphSAGE	1.182±0.024	2.095±0.013	0.738±0.018	1.338	86.119±0.240	83.330±0.123	97.459±0.041	73.631±1.765	73.700±0.980	76.136±0.155	**81.729**
	GAT	1.828±0.013	2.871±0.009	1.106±0.038	1.935	78.101±0.290	80.959±0.085	97.935±0.029	67.275±1.826	67.516±2.329	71.159±0.589	**77.157**
	GIN	1.138±0.009	1.223±0.051	0.717±0.010	**1.026**	86.721±0.628	84.474±0.120	97.448±0.029	81.177±1.469	77.002±0.135	76.554±0.390	**83.896**
	Avg.	**1.341**	2.058	0.833		**84.288**	**82.641**	**97.558**	74.002	73.550	**74.974**	

5 Conclusion

We have introduced a novel low-rank graph adapter that combines normalized graph convolution with trainable low-rank weight matrices, achieving performance competitive with SOTA fine-tuning methods for GNNs, including MPNNs and GTs. Our theoretical analysis highlights two key advantages over other concurrent methods: our adapter is sensitive to the input graph's structure (allowing it to capture second-hop interactions) and maintains a stable Lipschitz constant for graphs with sufficient connectivity, as described in Theorem 1. As the parameter count of GNN models grows and with the advent of graph foundation models, we expect fine-tuning methods to become more prevalent in graph representation learning. We hope our framework serves as a first step toward developing reliable methods for this purpose.

Table 2: Transductive Learning: Node Classification on Citation Networks. Best, second-best, and third-best accuracy (mean ± standard dev) in red, blue, and brown.

Tuning Method	Pre-trained Model	Accuracy (↑)		
		Cora	Citeseer	PubMed
Full Fine-tuning	NodeFormer	85.180±0.847	77.071±1.666	87.897±0.150
	DIFFormer-s	83.187±5.035	79.254±4.812	89.943±3.721
Surgical Fine-tuning	NodeFormer	78.336±1.328	75.510±2.220	87.809±0.474
	DIFFormer-s	81.798±5.335	77.965±4.923	87.632±3.845
BitFit	NodeFormer	78.237±0.921	74.630±2.410	87.857±0.564
	DIFFormer-s	81.418±5.481	77.589±5.102	87.247±3.956
LoRA	NodeFormer	85.327±0.766	76.230±1.424	87.492±0.083
	DIFFormer-s	78.943±6.162	75.210±5.843	84.676±4.275
Adapter	NodeFormer	78.582±1.343	74.590±2.605	87.945±0.502
	DIFFormer-s	81.403±5.380	77.572±5.015	87.234±3.862
G-Adapter	NodeFormer	83.653±0.707	74.390±2.257	89.094±0.292
	DIFFormer-s	67.897±17.506	64.523±16.842	73.271±15.378
AdapterGNN	NodeFormer	78.730±1.206	74.910±2.095	88.283±0.531
	DIFFormer-s	80.093±6.692	76.288±5.987	85.898±4.523
GConv-Adapter (ours)	NodeFormer	81.339±0.982	76.631±2.014	89.162±0.342
	DIFFormer-s	82.841±5.023	78.967±4.756	88.615±3.589

ETHICS STATEMENT

We believe that the potential societal consequences are minimal and do not require specific highlighting at this time.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7319–7328, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.568. URL https://aclanthology.org/2021.acl-long.568.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1.

Haitz Sáez de Ocáriz Borde, Artem Lukoianov, Anastasis Kratsios, Michael Bronstein, and Xiaowen Dong. Scalable message passing neural networks: No need for attention in large graph representation learning, 2024. URL https://arxiv.org/abs/2411.00835.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021. URL https://arxiv.org/abs/2104.13478.

Richard A. Brualdi, Ángeles Carmona, P. van den Driessche, Stephen Kirkland, and Dragan Stevanović. *Combinatorial matrix theory*. Advanced Courses in Mathematics. CRM Barcelona. Birkhäuser/Springer, Cham, 2018. ISBN 978-3-319-70952-9; 978-3-319-70953-6. doi: 10.1007/978-3-319-70953-6. URL https://doi.org/10.1007/978-3-319-70953-6. Notes of the lectures delivered at Centre de Recerca Matemàtica (CRM), Bellaterra, June 29–July 3, 2015.

Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. Parameter-efficient fine-tuning design spaces. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XSRSWxyJIC.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models, 2022. URL https://arxiv.org/abs/2203.06904.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Chin-Lun Fu, Zih-Ching Chen, Yun-Ru Lee, and Hung-yi Lee. AdapterBias: Parameter-efficient token-dependent representation shift for adapters in NLP tasks. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 2608–2621, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.199. URL https://aclanthology.org/2022.findings-naacl.199.

Anna Gaulton, Louisa J. Bellis, A. Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, et al. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research*, 40(D1):D1100–D1107, 2011.

Anchun Gui, Jinqiang Ye, and Han Xiao. G-adapter: Towards structure-aware parameter-efficient transfer learning for graph transformer networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 12226–12234, 2024.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey, 2024. URL https://arxiv.org/abs/2403.14608.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=0RDcd5Axok.

Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, Cambridge, 1990. ISBN 0-521-38632-2. Corrected reprint of the 1985 original.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Shengding Hu, Zhen Zhang, Ning Ding, Yadao Wang, Yasheng Wang, Zhiyuan Liu, and Maosong Sun. Sparse structure search for delta tuning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022b. URL https://openreview.net/forum?id=oOte_397Q4P.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020a.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=HJlWWJSFDH.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020c. URL https://openreview.net/forum?id=HJlWWJSFDH.

Zeyinzi Jiang, Chaojie Mao, Ziyuan Huang, Yiliang Lv, Deli Zhao, and Jingren Zhou. Rethinking efficient tuning methods from a unified perspective, 2023. URL https://arxiv.org/abs/2303.00690.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Annual Meeting of the Association for Computational Linguistics*, 2021.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=APuPRxjHvZ.

Shengrui Li, Xueting Han, and Jing Bai. Adaptergnn: Parameter-efficient fine-tuning improves generalization in gnns. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:13600–13608, 03 2024. doi: 10.1609/aaai.v38i12.29264.

Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S. Yu, and Chuan Shi. Towards graph foundation models: A survey and beyond. *ArXiv*, abs/2310.11829, 2023. URL https://api.semanticscholar.org/CorpusID:264288909.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=bqGK5PyI6-N.

Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph foundation models are already here. In *International Conference on Machine Learning*, 2024. URL https://api.semanticscholar.org/CorpusID:267412744.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. Unipelt: A unified framework for parameter-efficient language model tuning. 2022.

Andreas Mayr, Günter Klambauer, Thomas Unterthiner, Marvin Steijaert, Jörg K. Wegner, Hugo Ceulemans, Djork-Arné Clevert, and Sepp Hochreiter. Large-scale comparison of machine learning methods for drug target prediction on chembl. *Chemical Science*, 9(24):5441–5451, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. AdapterHub: A framework for adapting transformers. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 46–54, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.7. URL https://aclanthology.org/2020.emnlp-demos.7.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.39. URL https://aclanthology.org/2021.eacl-main.39.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. *Advances in Neural Information Processing Systems*, 35, 2022.

Bharath Ramsundar, Peter Eastman, Patrick Walters, Vijay Pande, Karl Leswing, and Zhenqin Wu. *Deep Learning for the Life Sciences*. O'Reilly Media, 2019. https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. AdapterDrop: On the efficiency of adapters in transformers. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7930–7946, Online

and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.626. URL https://aclanthology.org/2021.emnlp-main.626.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008. doi: 10.1609/aimag.v29i3.2157. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2157.

Teague Sterling and John J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015. doi: 10.1021/acs.jcim.5b00559.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Ruize Wang, Duyu Tang, Nan Duan, zhongyu wei, Xuanjing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou. K-adapter: Infusing knowledge into pre-trained models with adapters, 2021. URL https://openreview.net/forum?id=CLnj31GZ4cI.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5744–5760, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.388. URL https://aclanthology.org/2022.emnlp-main.388.

Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. Difformer: Scalable (graph) transformers induced by energy constrained diffusion. In *International Conference on Learning Representations (ICLR)*, 2023.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1810.00826.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=OeWooOxFwDa.

Bruce X.B. Yu, Jianlong Chang, Lingbo Liu, Qi Tian, and Chang Wen Chen. Towards a unified view on visual parameter-efficient transfer learning. *arXiv preprint arXiv:2210.00788*, 2022.

Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Haitz Sáez de Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. Asymmetry in low-rank adapters of foundation models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=txRZBD8tBV.

A PROOFS

Proof of Proposition 1. By definition of the Adapter GNN, in equation 1, we directly see that

$$A_{\text{AdapterGNN}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}) = \text{BN}\left(\sigma\left(\mathbf{X}\mathbf{W}_{\text{down}}\right)\mathbf{W}_{\text{up}}\right) \tag{8}$$

$$= A_{\text{AdapterGNN}}(\mathbf{X}|\mathcal{G}', \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}), \tag{9}$$

which completes our proof. □

Proof of Proposition 2. Let $N, D, r \in \mathbb{N}_+$, set $r = N$, and $D = 1$ (we henceforth identify matrices in $\mathbb{R}^{N\times 1}$ with vectors in $\mathbb{R}^N$). Let $\mathcal{G}$ be any δ-regular graph on N vertices. Let $\mathbf{W}_{down} = \mathbf{W}_{up} \overset{\text{def.}}{=} I_N$ (identity), and $\sigma = \text{ReLU}$. We would like a lower bound on the Lipschitz constant $L \overset{\text{def.}}{=} \text{Lip}\left(A_{\text{G-Adapter}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up})\right)$ of $A_{\text{G-Adapter}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up})$

$$\begin{aligned} L &\overset{\text{def.}}{=} \sup_{\substack{\mathbf{X}\neq\mathbf{X}' \\ \mathbf{X},\mathbf{X}'\in\mathbb{R}^{N\times D}}} \frac{\left\|A_{\text{G-Adapter}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) - A_{\text{G-Adapter}}(\mathbf{X}'|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up})\right\|}{\|\mathbf{X} - \mathbf{X}'\|} \\ &\geq \frac{\left\|A_{\text{G-Adapter}}(\mathbf{1}_N|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) - A_{\text{G-Adapter}}(\mathbf{0}_N|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up})\right\|}{\|\mathbf{X} - \mathbf{X}'\|}, \end{aligned} \tag{10}$$

where $\mathbf{1}_N$ is the vector in $\mathbb{R}^N$ with all entries equal to 1 and $\mathbf{0}_N$ is the zero vector in $\mathbb{R}^N$.

Next, note that the adjacency matrix $\mathbf{A}$ is such that: each row of $\mathbf{A}$ has exactly δ entries equal to 1 and all other entries equal to 0. Therefore, by definition of the G-Adapted (ignoring layer norms), in equation 3, we have that:

$$A_{\text{G-Adapter}}(\mathbf{0}_N|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) = \mathbf{0}_N + \sigma\left(\tilde{\mathbf{A}}\,\mathbf{0}_N\mathbf{W}_{\text{down}}\,\mathbf{W}_{\text{up}}\right) = \mathbf{0}_N. \tag{11}$$

In contrast, when processing $\mathbf{1}_N$ we have, considering that $\tilde{\mathbf{A}} = \mathbf{A} + I_N$:

$$A_{\text{G-Adapter}}(\mathbf{1}_N|\mathcal{G}, \mathbf{W}_{down}, \mathbf{W}_{up}) = \mathbf{1}_N + \text{ReLU}\left(\tilde{\mathbf{A}}\,\mathbf{1}_N I_N I_N\right) \tag{12}$$

$$= \mathbf{1}_N + \text{ReLU}\left((1+\delta)\mathbf{1}_N\right) \tag{13}$$

$$= \mathbf{1}_N + (1+\delta)\mathbf{1}_N \tag{14}$$

$$= (2+\delta)\mathbf{1}_N. \tag{15}$$

Upon combining the computations in equation 12-equation 15 with those in equation 11, and with the lower-bound on the Lipschitz constant L of the G-Adapter in equation 10, we find that

$$L \geq \frac{\left\|A_{\text{G-Adapter}}(\mathbf{1}_N|G, \mathbf{W}_{down}, \mathbf{W}_{up}) - A_{\text{G-Adapter}}(\mathbf{0}_N|G, \mathbf{W}_{down}, \mathbf{W}_{up})\right\|}{\|\mathbf{X} - \mathbf{X}'\|} \tag{16}$$

$$= \frac{\|\mathbf{0}_N - (2+\delta)\mathbf{1}_N\|}{\|\mathbf{0}_N - \mathbf{1}_N\|} \tag{17}$$

$$= \frac{(2+\delta)\|\mathbf{1}_N\|}{\|\mathbf{0}_N - \mathbf{1}_N\|} \tag{18}$$

$$= (2+\delta). \tag{19}$$

Thus, concluding our proof. □

In terms of its asymptotic behavior, in the particular case when $\delta = N - 1$ (the complete graph scenario), as $N \to \infty$ we have

$$\text{Lip}(f) \;\in\; \Theta(N).$$

That is, the Lipschitz constant diverges linearly with the number of vertices N.

Note that the proof above follows for any δ-regular graph, not just for expander graphs. δ-regular graphs are expanders with high probability. Even expanders, which are known to have favorable structural properties due to their strong connectivity and robust spectral properties, can lead to a diverging Lipschitz constant when the unnormalized adjacency matrix is used.

Proof of Proposition 3. To establish the negation of the graph-agnosticism of the GConv-Adapter, it is enough to exhibit a single case where equation 6 holds. To this end, let $r = N = D$, $\alpha = 1$, $\mathbf{W}_{down} \overset{\text{def.}}{=} \mathbf{W}_{up} = I_2$, and $\mathbf{X} = (1, 0) \in \mathbb{R}^2$. Let $\mathcal{G} = (\{0, 1\}, \emptyset)$ be the graph on two vertices with no edges and $\mathcal{G}' = (\{0, 1\}, \{(0, 1)\}) = K_2$ be the complete graph on two vertices. Under these specification, for both $\tilde{\mathcal{G}} \in \{\mathcal{G}, \mathcal{G}'\}$ we have

$$A_{\text{G-Conv}}(\mathbf{X}|\tilde{\mathcal{G}}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}) = \mathbf{X} + \left(\hat{\mathbf{A}}^{\tilde{\mathcal{G}}}\ \text{ReLU}(\hat{\mathbf{A}}^{\tilde{\mathcal{G}}}\,\mathbf{X})\right), \tag{20}$$

where $\hat{\mathbf{A}}^{\tilde{\mathcal{G}}}$ denotes the normalized adjacency matrix of the graph $\tilde{\mathcal{G}}$.

Let us simplify the right-hand side of equation 20 in the case where $\tilde{\mathcal{G}} = \mathcal{G}$. First, we note that the degree matrix of $\mathcal{G}$ (with self-loops) is $\tilde{D}^{\mathcal{G}}_{i,i} = 1$, i.e. $\tilde{\mathbf{D}}^{\mathcal{G}} = I_2$, and its adjacency matrix $\mathbf{A}^{\mathcal{G}}$ is the 2×2 zero matrix $\mathbf{0}_2$. Whence, its normalized adjacency matrix $\hat{\mathbf{A}}^{\mathcal{G}}$ is

$$\hat{\mathbf{A}}^{\mathcal{G}} = I_2^{-1/2}(I_2 + \mathbf{0}_2)I_2^{-1/2} = I_2. \tag{21}$$

Using the fact that $\text{ReLU}(x) = x$ for all $x \geq 0$, equation 20 and equation 21 imply that

$$\begin{aligned} A_{\text{G-Conv}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}) &= \mathbf{X} + \left(\hat{\mathbf{A}}^{\mathcal{G}}\ \text{ReLU}(\hat{\mathbf{A}}^{\mathcal{G}}\,\mathbf{X})\right) && (22)\\ &= (1, 0)^\top + \left(\text{ReLU}(1, 0)^\top\right) && (23)\\ &= (2, 0)^\top. && (24) \end{aligned}$$

Now, in the case where $\tilde{\mathcal{G}} = \mathcal{G}' = K_2$, we see that the degree matrix with self-loops $\tilde{\mathbf{D}}^{\mathcal{G}'} = 2I_2$, while this time, the adjacency matrix $\mathbf{A}^{\mathcal{G}'}$ of $\mathcal{G}'$ is the permutation matrix

$$\mathbf{A}^{\mathcal{G}'} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Consequently, the normalized adjacency matrix of $\mathcal{G}'$ is

$$\hat{\mathbf{A}}^{\mathcal{G}'} = \frac{1}{\sqrt{2}} I_2 (I_2 + \mathbf{A}^{\mathcal{G}'}) \frac{1}{\sqrt{2}} I_2 = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}. \tag{25}$$

Now, incorporating equation 25 into the right-hand side of equation 20, and again using the fact that the ReLU activation function acts as the identity on $[0, \infty)$ we find that

$$\begin{aligned} A_{\text{G-Conv}}(\mathbf{X}|\tilde{\mathcal{G}}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}) &= (1, 0)^\top + \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \text{ReLU}\left(\frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} (1, 0)^\top\right) && (26)\\ &= (1, 0)^\top + \frac{1}{4} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} (1, 1)^\top && (27)\\ &= (1, 0)^\top + \frac{1}{4}(2, 2)^\top && (28)\\ &= (1.5, 0.5)^\top. && (29) \end{aligned}$$

Comparing the computations in equation 22-equation 24 with those in equation 26-equation 29 yields the conclusion. □

Proof of Proposition 1.
Let $\mathcal{G}$ be any δ-regular graph (including self-loops) and let $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\mathcal{G}}$ be its respective symmetrically normalized adjacency matrix. We also have $\mathbf{W}_{\text{down}} \in \mathbb{R}^{D \times r}$, $\mathbf{W}_{\text{up}} \in \mathbb{R}^{r \times D}$, the low rank learnable matrices. Let us upper-bound the Lipschitz constant of the $A_{\text{G-Conv}}(\cdot|\mathcal{G}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}})$. Thus, we fix $\mathbf{X}, \mathbf{X}' \in \mathbb{R}^{N \times D}$ and we compute

$$\begin{aligned} &\left\| A_{\text{G-Conv}}(\mathbf{X}|\mathcal{G}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}) - A_{\text{G-Conv}}(\mathbf{X}'|\mathcal{G}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}}) \right\| && (30)\\ \leq &\Big\| \mathbf{X} + \alpha \cdot \text{Normalization}\left(\hat{\mathbf{A}}\, \sigma\left(\hat{\mathbf{A}}\, \mathbf{X}\, \mathbf{W}_{\text{down}}\right) \mathbf{W}_{\text{up}}\right) && (31)\\ &- \mathbf{X}' + \alpha \cdot \text{Normalization}\left(\hat{\mathbf{A}}\, \sigma\left(\hat{\mathbf{A}}\, \mathbf{X}'\, \mathbf{W}_{\text{down}}\right) \mathbf{W}_{\text{up}}\right) \Big\| && (32) \end{aligned}$$

$$\leq \|\mathbf{X} - \mathbf{X}'\| + \alpha \Big\| \text{Normalization}\left(\hat{\mathbf{A}}\,\sigma\left(\hat{\mathbf{A}}\,\mathbf{X}\,\mathbf{W}_{\text{down}}\right)\mathbf{W}_{\text{up}}\right) \tag{33}$$

$$- \text{Normalization}\left(\hat{\mathbf{A}}\,\sigma\left(\hat{\mathbf{A}}\,\mathbf{X}'\,\mathbf{W}_{\text{down}}\right)\mathbf{W}_{\text{up}}\right)\Big\| \tag{34}$$

$$\leq \|\mathbf{X} - \mathbf{X}'\| + \alpha \Big\| \left(\hat{\mathbf{A}}\,\sigma\left(\hat{\mathbf{A}}\,\mathbf{X}\,\mathbf{W}_{\text{down}}\right)\mathbf{W}_{\text{up}}\right) - \left(\hat{\mathbf{A}}\,\sigma\left(\hat{\mathbf{A}}\,\mathbf{X}'\,\mathbf{W}_{\text{down}}\right)\mathbf{W}_{\text{up}}\right)\Big\| \tag{35}$$

$$\leq \|\mathbf{X} - \mathbf{X}'\| + \alpha \|\hat{\mathbf{A}}\|_{op}\, L_\sigma\, \|\hat{\mathbf{A}}\|_{op} \Big\| \mathbf{X}\,\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}} - \mathbf{X}'\,\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}\Big\| \tag{36}$$

$$\leq \|\mathbf{X} - \mathbf{X}'\| + \alpha \|\hat{\mathbf{A}}\|_{op}\, L_\sigma\, \|\hat{\mathbf{A}}\|_{op} \Big\| \mathbf{X} - \mathbf{X}'\Big\| \, \|\mathbf{W}_{\text{down}}\|_{op}\|\mathbf{W}_{\text{up}}\|_{op} \tag{37}$$

$$= \|\mathbf{X} - \mathbf{X}'\| + \|\hat{\mathbf{A}}\|_{op}^2 \alpha L_\sigma \|\mathbf{W}_{\text{down}}\|_{op}\|\mathbf{W}_{\text{up}}\|_{op} \Big\| \mathbf{X} - \mathbf{X}'\Big\| \tag{38}$$

$$\leq (1 + \|\hat{\mathbf{A}}\|_{op}^2 \alpha L_\sigma \|\mathbf{W}_{\text{down}}\|_{op}\|\mathbf{W}_{\text{up}}\|_{op}) \Big\| \mathbf{X} - \mathbf{X}'\Big\|. \tag{39}$$

Consequentially, $\text{Lip}\left(A_{\text{G-Conv}}(\cdot|G, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}})\right)$ is Lipschitz with constant at-most

$$\text{Lip}\left(A_{\text{G-Conv}}(\cdot|G, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{up}})\right) \leq 1 + \|\hat{\mathbf{A}}\|_{op}^2 \alpha L_\sigma \|\mathbf{W}_{\text{down}}\|_{op}\|\mathbf{W}_{\text{up}}\|_{op}. \tag{40}$$

If $\mathcal{G}$ is δ-regular, then $\tilde{D}_{i,i} = \tilde{D}_{j,j} = \delta + 1$ for every $i, j \in \{1, \ldots, N\}$; otherwise, $\tilde{D}_{i,i} \geq 1 + \deg_-(\mathcal{G})$, where $\deg_-(\mathcal{G})$ refers to the minimal degree of $\mathcal{G}$. In particular, when $\mathcal{G}$ is δ-regular, $\tilde{\mathbf{D}}$ equals to the scalar matrix $\tilde{\mathbf{D}} = (\delta + 1)I_N$ and otherwise, it is a diagonal matrix. Whence, the operator norm of $\hat{\mathbf{A}}$ reduces to

$$\|\hat{\mathbf{A}}\|_{op} = \big\|\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\big\|_{op} \tag{41}$$

$$\leq \big\|((1 + \deg_-(\mathcal{G}))I_N)^{-1/2}\tilde{\mathbf{A}}((1 + \deg_-(\mathcal{G}))I_N)^{-1/2}\big\|_{op} \tag{42}$$

$$= \frac{1}{\sqrt{(1 + \deg_-(\mathcal{G}))}}\big\|\tilde{\mathbf{A}}\big\|_{op}\frac{1}{\sqrt{(1 + \deg_-(\mathcal{G}))}} \tag{43}$$

$$\leq \frac{1}{(1 + \deg_-(\mathcal{G}))}\big(1 + \|\mathbf{A}\|_{op}\big) \tag{44}$$

where $\mathbf{A}$ is the adjacency matrix of $\mathcal{G}$ without self-loops. The last inequality is based on the triangle inequality and on the fact that the operator norm of the identity matrix is 1. Also, since the adjacency matrix is symmetric, then, the Spectral Theorem, see e.g. (Horn & Johnson, 1990, Theorem 2.5.6), implies that all eigenvalues of $\mathbf{A}$ are real and, moreover, $\|\mathbf{A}\|_{op}$ equals to the absolute maximal eigenvalue of $\mathbf{A}$, i.e. the spectral radius $\rho(\mathbf{A})$ of $\mathbf{A}$. The δ-regularity of $\mathbf{A}$ implies that, by the simple spectral bound in (Brualdi et al., 2018, equation 3.3), we have

$$\rho(\mathbf{A}) \leq \deg_+(\mathcal{G}) \leq \begin{cases} \delta & \text{if } \mathcal{G} \text{ is } \delta\text{-regular} \\ 1 & \text{else.} \end{cases} \tag{45}$$

In the above, $\deg_+(\mathcal{G})$ denotes the maximal degree of $\mathcal{G}$. Furthermore, in the case where $\mathcal{G}$ is δ-regular the above inequalities become equalities by (Brualdi et al., 2018, Theorem 3.2.3). Thus, we find that

$$\begin{aligned} \|\hat{\mathbf{A}}\|_{op} &\leq \frac{1}{1 + \deg_-(\mathcal{G})}\big(1 + \|\mathbf{A}\|_{op}\big) = \frac{1}{1 + \deg_-(\mathcal{G})}\big(1 + \rho(\mathbf{A})\big) \\ &= \begin{cases} \frac{1+\deg_+(\mathcal{G})}{1+\deg_-(\mathcal{G})} = \frac{1+\delta}{1+\delta} = 1, & \text{if } \mathcal{G} \text{ is } \delta\text{-regular,} \\ \frac{1+\rho(\mathbf{A})}{1+\deg_-(\mathcal{G})}, & \text{otherwise.} \end{cases} \end{aligned} \tag{46}$$

Note that for arbitrary connected graphs with significant degree variability, the minimal degree can be much lower than the spectral radius: consider for instance star graphs.

Finally, the right-hand side of equation 40 yields the desired bound since

$$\text{Lip}\left(A_{\text{G-Conv}}\right) \leq 1 + \alpha L_\sigma \|\mathbf{W}_{\text{down}}\|_{op}\|\mathbf{W}_{\text{up}}\|_{op} \times \begin{cases} 1 & \text{if } \mathcal{G} \text{ is } \delta\text{-regular} \\ \frac{(1+\rho(\mathbf{A}))^2}{(1+\deg_-(\mathcal{G}))^2} & \text{else.} \end{cases} \tag{47}$$

□

Given that our analysis holds for any δ-regular graph, the Lipschitz bound applies to every δ-regular graph, including those produced randomly: the derivation did not depend on any additional assumptions regarding the topology of $\mathcal{G}$.

B Complexity Analysis

We provide a detailed comparison of the time and space complexity of GConv-Adapter against other graph-specific PEFT methods, namely AdapterGNN (Li et al., 2024) and G-Adapter (Gui et al., 2024).

Computational Complexity Overview. Let $|\mathcal{V}| = N$ nodes and $|\mathcal{E}| = E$ edges, and let D be the input feature dimension. The GConv-Adapter consists of two sparse-dense multiplications with the normalized adjacency matrix $\hat{\mathbf{A}}$ and two dense linear projections via the low-rank matrices $\mathbf{W}_{down} \in \mathbb{R}^{D\times r}$ and $\mathbf{W}_{up} \in \mathbb{R}^{r\times D}$. This yields an overall per-layer time complexity of $\mathcal{O}(ED + Er + NDr)$. In practice, since $r \ll D$, the $\mathcal{O}(Er)$ term becomes negligible compared to the $\mathcal{O}(ED)$ term. As a result, the overall complexity simplifies to $\mathcal{O}(ED + NDr)$, where the first term accounts for the sparse graph convolutions and the second captures the cost of the dense low-rank projections. The space complexity is dominated by the trainable parameters, which scale as $\mathcal{O}(Dr)$, significantly smaller than the $\mathcal{O}(D^2)$ complexity incurred by full fine-tuning. This confirms that GConv-Adapter remains lightweight and scalable even for large-scale graphs. We provide detailed matrix-level derivations of the time and space complexity next.

B.1 Time Complexity

Table 3 summarizes the time complexity of the core operations in each method. For GConv-Adapter, the cost arises from two rounds of sparse graph convolution, followed by low-rank projections and normalization. G-Adapter performs a single sparse aggregation before projection. AdapterGNN, by contrast, does not involve any graph-aware operation within the adapter itself.

Table 3: Time complexity of graph-specific PEFT methods

Operation	**AdapterGNN**	**G-Adapter**	**GConv-Adapter**
Graph Convolution	–	$\mathcal{O}(ED)$	$\mathcal{O}(ED + Er)$
Down-Projection	$\mathcal{O}(NDr)$	$\mathcal{O}(NDr)$	$\mathcal{O}(NDr)$
Up-Projection	$\mathcal{O}(NDr)$	$\mathcal{O}(NDr)$	$\mathcal{O}(NDr)$
Activation	$\mathcal{O}(Nr)$	$\mathcal{O}(ND)$	$\mathcal{O}(Nr)$
Normalization	$\mathcal{O}(ND)$	$\mathcal{O}(ND)$	$\mathcal{O}(ND)$
Total	$\mathcal{O}(NDr)$	$\mathcal{O}(ED + NDr)$	$\mathcal{O}(ED + Er + NDr)$

In practice, since $r \ll D$, the $\mathcal{O}(Er)$ term in GConv-Adapter is dominated by the $\mathcal{O}(ED)$ term. This allows us to simplify the total complexity to $\mathcal{O}(ED + NDr)$, where the first term accounts for sparse graph convolutions and the second for dense low-rank projections.

B.2 Space Complexity

Table 4 compares the number of additional parameters introduced by each method. All methods rely on a pair of projection matrices for dimensionality reduction and expansion, with optional normalization and scaling parameters. GConv-Adapter retains the same parameter efficiency as AdapterGNN and G-Adapter while adding structural awareness.

Here, α denotes the learnable scaling factor (where applicable), while γ and β are the scale and shift parameters of the normalization layers. Despite its added structural processing, GConv-Adapter preserves the low space footprint characteristic of PEFT methods.

Table 4: Space complexity of graph-specific PEFT methods

Method	Learned Parameters	Total Space Complexity
AdapterGNN	$\mathbf{W}_{\text{down}} \in \mathbb{R}^{D\times r}, \mathbf{W}_{\text{up}} \in \mathbb{R}^{r\times D}, \alpha, \gamma, \beta$	$\mathcal{O}(Dr)$
G-Adapter	$\mathbf{W}_{\text{down}} \in \mathbb{R}^{D\times r}, \mathbf{W}_{\text{up}} \in \mathbb{R}^{r\times D}, \gamma, \beta$	$\mathcal{O}(Dr)$
GConv-Adapter	$\mathbf{W}_{\text{down}} \in \mathbb{R}^{D\times r}, \mathbf{W}_{\text{up}} \in \mathbb{R}^{r\times D}, \alpha, \gamma, \beta$	$\mathcal{O}(Dr)$

C Implementation Details

Datasets and Evaluation Protocol For the inductive setting, we use nine molecular property prediction datasets (Ramsundar et al., 2019): ESOL, FreeSolv, Lipophilicity, Tox21, SIDER, ClinTox, BACE, MUV, and HIV. Among these, ESOL (1.1K graphs), FreeSolv (0.6K), BACE (1.5K), ClinTox (1.4K), and SIDER (1.4K) are considered small-scale. Lipophilicity (4.2K) and Tox21 (7.8K) represent medium-scale datasets, while HIV (41K) and MUV (93K) are large-scale datasets. Consistent with prior work (Hu et al., 2020b), we use a scaffold split for HIV and random splits for the others. RMSE is used to evaluate ESOL, FreeSolv, and Lipophilicity, while ROC-AUC is used for the remaining datasets. For the transductive setting, we evaluate our approach on three standard node classification datasets: Cora, Citeseer, and PubMed (Sen et al., 2008). These datasets consist of citation networks, where nodes represent documents and edges correspond to citation links. We use a standard random 50%/25%/25% split for train/val/test.

Experimental Setup All experiments were implemented using PyTorch (Paszke et al., 2019) and PyG (Fey & Lenssen, 2019) on NVIDIA A100 GPUs. For the inductive setting, we maintained consistency with the hyperparameters and training strategies used in previous work by Hu et al. (2020c), ensuring reproducibility and comparability. Our results were obtained without any special hyperparameter tuning for individual datasets. In the transductive setting, we used the same hyperparameters for fine-tuning as those applied during pre-training. The primary difference was that the fine-tuning process was limited to 300 epochs, with early stopping triggered if the validation loss did not improve for 20 consecutive epochs. The bottleneck dimension for all low-rank based PEFT methods (LoRA, Adapter, AdapterGNN, G-Adapter, and GConv-Adapter) was set to 16 to maintain consistency across experiments.

Pre-Trained Models: Inductive For the inductive learning experiments, we utilized pre-trained models of four widely-adopted GNN architectures—GCN Kipf & Welling (2017), GraphSAGE Hamilton et al. (2017), GAT Veličković et al. (2018), and GIN Xu et al. (2018). We applied a two-stage pre-training process: self-supervised node-level pre-training, followed by supervised graph-level property prediction pre-training, as in Hu et al. (2020c). Specifically, for node-level pre-training, 2 million unlabeled molecules sampled from the ZINC15 database (Sterling & Irwin, 2015) were used, while graph-level pre-training was conducted using the ChEMBL dataset (Mayr et al., 2018; Gaulton et al., 2011), which contains 456K molecules and 1,310 diverse biochemical assays. Pre-trained checkpoints from Hu et al. (2020c) were employed for all experiments, using context prediction as the node-level pre-training strategy. This combination of context prediction and graph-level supervised pre-training has been shown to mitigate the risk of negative transfer, thereby making the models highly suitable for fine-tuning on downstream molecular property prediction tasks.

Pre-Trained Models: Transductive For the transductive learning experiments, we utilized the NodeFormer (Wu et al., 2022) and DIFFormer-s (Wu et al., 2023) GT architectures. Both models were pre-trained using the ogbn-Arxiv dataset from the Open Graph Benchmark (OGB) Hu et al. (2020a), a large-scale citation network of computer science arXiv papers, which contains over 169,000 nodes (papers) and more than 1.1 million edges (citation links). The choice of ogbn-Arxiv for pre-training was motivated by its domain similarity to the target datasets (Cora, Citeseer, and PubMed). Pre-training on this large and structurally rich dataset allows the models to learn effective node representations and capture important structural patterns in citation networks, improving their ability to transfer knowledge when fine-tuned on the downstream tasks. In the case of NodeFormer, the model was configured with 64 hidden channels, 5 layers, and 1 attention head, utilizing an identity transformation for the relational bias with a regularization weight of 0.1. The training

process incorporated gumbel-softmax sampling for efficient message passing, batch normalization, and residual connections. A learning rate of 0.01, no weight decay, and a batch size of 10,000 were applied during the 1,000 epochs of training. Similarly, DIFFormer-s was pre-trained with 64 hidden channels and 5 layers, using 1 attention head and an alpha value of 0.5 to balance the residual connections. Batch normalization and residual connections were enabled, along with the use of graph positional embeddings. The model employed a simple kernel, where queries and keys were normalized, and attention was computed using dot products between these normalized inputs. A dropout rate of 0.2 and a weight decay of 0.01 were applied to regularize the training, with a learning rate of 0.001, a batch size of 10,000, and training for 1,000 epochs. For both NodeFormer and DIFFormer-s, all other hyperparameters and architectural choices were kept consistent with their original implementations as described in the respective papers (Wu et al., 2022; 2023).

Baselines As baselines, we include full fine-tuning and four commonly used PEFT methods: Surgical Fine-tuning (Lee et al., 2023), BitFit (Ben Zaken et al., 2022), LoRA (Hu et al., 2022a), and Adapter (Houlsby et al., 2019). Additionally, we evaluate two graph-specific PEFT approaches: G-Adapter and AdapterGNN, which are extensively discussed in the main text.

D Overall Performance Comparison

This appendix complements the discussion in Section 4. Here, we analyze average results from the perspectives of both architectures and tasks.

Comparison across architectures First we compare average results for different architectures. In regression tasks (Table 5), GConv-Adapter demonstrates particularly strong performance with the GIN architecture, achieving the best RMSE of 1.026 among all methods and architectures. Compared to full fine-tuning, GConv-Adapter shows substantial improvements across the different architectures. With GIN, it improves upon full fine-tuning's RMSE of 1.501 by 31.64%. For GraphSAGE and GAT, while not achieving the best performance, GConv-Adapter still outperforms full fine-tuning by 5.77% and 8.64% respectively (1.338 vs 1.420 and 1.935 vs 2.118). Only with GCN does GConv-Adapter fall behind full fine-tuning (1.343 vs 1.195). When compared to other graph-specific PEFT methods in regression tasks, GConv-Adapter shows competitive performance. AdapterGNN achieves the best results on three out of four architectures (GCN: 1.064, GraphSAGE: 1.172, GAT: 1.740), leading to the best overall average of 1.291. G-Adapter, with an average RMSE of 1.526, performs better than GConv-Adapter on GCN (1.267 vs 1.343) but falls behind on other architectures. Despite not achieving the best average RMSE (1.411), GConv-Adapter's strong performance with GIN and consistent improvements over full fine-tuning demonstrate its effectiveness as the second-best overall PEFT method. Compared to classical PEFT methods, GConv-Adapter shows clear advantages. It outperforms the best classical PEFT method, Adapter (average RMSE: 1.625), by 13.16%. Other classical methods like LoRA (1.645), Surgical Fine-tuning (1.913), and BitFit (2.870) show significantly higher RMSE values across all architectures.

Table 5: Table comparing overall performance on molecular prediction regression tasks (ESOL, FreeSolv, and Lipophilicity), with results reported as the average Root Mean Square Error (RMSE) across all tasks (lower RMSE indicates better performance). The best result for each pre-trained model is highlighted in **bold**.

Method	GCN	GraphSAGE	GAT	GIN	Average
Full Fine-tuning	1.195	1.420	2.118	1.501	1.559
Surgical Fine-tuning	1.854	1.798	2.185	1.816	1.913
BitFit	2.887	2.788	2.981	2.824	2.870
LoRA	1.547	1.422	2.055	1.556	1.645
Adapter	1.155	1.525	2.148	1.672	1.625
G-Adapter	1.267	1.305	2.027	1.505	1.526
AdapterGNN	**1.064**	**1.172**	**1.740**	1.187	**1.291**
GConv-Adapter (ours)	1.343	1.338	1.935	**1.026**	1.411

In classification tasks (Table 6), GConv-Adapter demonstrates superior performance across multiple architectures. It achieves the best ROC-AUC scores with three out of four architectures: GraphSAGE (81.729%), GAT (77.157%), and GIN (83.896%). Compared to full fine-tuning, these results represent improvements of 2.7%, 2.95%, and 2.11% respectively. With GCN, GConv-Adapter

Table 6: Overall performance on molecular classification tasks (Tox21, SIDER, ClinTox, BACE, MUV, and HIV), with results reported as the average ROC-AUC across all tasks (higher ROC-AUC indicates better performance). The best result for each pre-trained model is highlighted in **bold**.

Method	GCN	GraphSAGE	GAT	GIN	Average
Full Fine-tuning	80.728	79.579	74.940	82.162	79.352
Surgical Fine-tuning	77.940	76.691	74.037	76.413	76.270
BitFit	66.934	69.615	68.645	66.974	68.042
LoRA	76.171	77.837	72.856	80.749	76.903
Adapter	80.160	77.395	73.701	80.850	78.027
G-Adapter	80.969	80.876	72.527	80.582	78.739
AdapterGNN	**82.170**	81.339	76.364	82.082	80.489
GConv-Adapter (ours)	81.786	**81.729**	**77.157**	**83.896**	**81.142**

(81.786%) comes second to the best performance achieved by AdapterGNN (82.170%), while still outperforms full fine-tuning by 1.31%. Against other graph-specific PEFT methods in classification tasks, GConv-Adapter shows consistent advantages. It surpasses G-Adapter's average performance (78.739%) by 3.05% and AdapterGNN's average (80.489%) by 0.81%, achieving the best overall average ROC-AUC of 81.142%. This superior performance is particularly evident with GAT and GIN architectures, where GConv-Adapter outperforms both G-Adapter and AdapterGNN by significant margins. When compared to classical PEFT methods in classification tasks, GConv-Adapter demonstrates substantial improvements. It outperforms all classical approaches, including Adapter (78.027%), LoRA (76.903%), Surgical Fine-tuning (76.270%), and BitFit (68.042%). The improvement margins range from 3.99% over Adapter to 19.25% over BitFit.

Comparison across tasks We present a detailed analysis of the performance of GConv-Adapter and other PEFT methods on regression and classification tasks, averaged across different architectural choices. Table 7 summarizes the average results for each fine-tuning method across the ESOL, FreeSolv, and Lipophilicity datasets.

Table 7: Average performance on molecular regression tasks (ESOL, FreeSolv, and Lipophilicity). The results are presented as average Root Mean Square Error (RMSE), with lower values indicating better performance. Each method's performance is averaged across all pre-trained models for each dataset, with the best result for each dataset highlighted in **bold**.

Method	ESOL	FreeSolv	Lipophilicity	Average
Full Fine-tuning	1.393	2.407	0.876	1.559
Surgical Fine-tuning	1.574	3.123	1.043	1.913
BitFit	2.316	5.096	1.198	2.870
LoRA	1.482	2.603	0.851	1.645
Adapter	1.414	2.598	0.862	1.625
G-Adapter	1.374	2.373	0.835	1.527
AdapterGNN	1.405	**1.688**	**0.780**	**1.291**
GConv-Adapter (ours)	**1.341**	2.058	0.833	1.411

GConv-Adapter demonstrates consistent improvements over full fine-tuning across all regression tasks. On the ESOL dataset, GConv-Adapter achieves an RMSE of 1.341, outperforming full fine-tuning (1.393) by 3.73%. For the FreeSolv dataset, GConv-Adapter (2.058) shows a substantial improvement of 14.49% over full fine-tuning (2.407). Similarly, on the Lipophilicity dataset, GConv-Adapter (0.833) improves upon full fine-tuning (0.876) by 4.90%. These consistent improvements across all datasets, with an average RMSE of 1.411 compared to full fine-tuning's 1.559 (9.49% improvement), demonstrate GConv-Adapter's effectiveness in adapting pre-trained models for molecular property regression tasks. When compared to other graph-specific PEFT methods, GConv-Adapter shows competitive performance. On the ESOL dataset, GConv-Adapter (1.341) outperforms both G-Adapter (1.374) and AdapterGNN (1.405) by 2.40% and 4.55% respectively. For the FreeSolv dataset, while AdapterGNN achieves the best performance (1.688), GConv-Adapter (2.058) still outperforms G-Adapter (2.373) by 13.27%. On the Lipophilicity dataset, GConv-Adapter (0.833) performs slightly better than G-Adapter (0.835) but trails behind AdapterGNN (0.780) by 6.79%. Overall, while AdapterGNN shows the best average performance (1.291), GConv-Adapter's average RMSE of 1.411 represents strong performance, outperforming G-Adapter (1.527) by 7.59%. Among classical PEFT methods, Adapter shows the strongest perfor-

mance, with an average RMSE of 1.625. However, GConv-Adapter outperforms all classical PEFT methods across all datasets, showing improvements ranging from 13.16% over Adapter to 50.83% over BitFit.

Lastly, we provide a comprehensive evaluation of GConv-Adapter's performance, along with the other PEFT techniques, in the classification tasks. Table 8 summarizes the average results for each fine-tuning method across the Tox21, SIDER, ClinTox, BACE, MUV, and HIV datasets.

Table 8: Average performance on molecular classification tasks (Tox21, SIDER, ClinTox, BACE, MUV, and HIV). Results are reported as average ROC-AUC, where higher values indicate better performance. Each method's performance is averaged across all pre-trained models for each dataset, with the best result for each dataset highlighted in **bold**. The evaluated methods include Full Fine-tuning, Surgical Fine-tuning, BitFit, LoRA, Adapter, G-Adapter, AdapterGNN, and GConv-Adapter (proposed method).

Method	Tox21	SIDER	ClinTox	BACE	MUV	HIV	Average
Full Fine-tuning	82.795	82.481	97.404	68.291	72.288	72.695	79.352
Surgical Fine-tuning	80.034	81.348	97.371	61.291	69.244	68.334	76.270
BitFit	68.587	79.159	96.611	53.272	52.722	57.903	68.042
LoRA	82.054	81.731	97.033	69.874	59.954	70.772	76.903
Adapter	82.728	82.152	97.067	67.574	69.098	69.541	78.027
G-Adapter	83.343	82.464	97.119	**74.054**	66.347	69.103	78.738
AdapterGNN	83.977	81.989	97.136	73.049	**73.799**	72.982	80.489
GConv-Adapter (ours)	**84.288**	**82.641**	**97.558**	74.002	73.550	**74.974**	**81.142**

Compared to full fine-tuning, GConv-Adapter demonstrates consistent improvements across most classification tasks. For Tox21, GConv-Adapter achieves a ROC-AUC of 84.288%, surpassing full fine-tuning (82.795%) by 1.80%. On SIDER, GConv-Adapter (82.641%) slightly outperforms full fine-tuning (82.481%) by 0.19%. The improvement is similar on ClinTox, where GConv-Adapter (97.558%) exceeds full fine-tuning (97.404%) by 0.15%. On BACE, GConv-Adapter shows substantial improvement with 74.002% compared to full fine-tuning's 68.291%, an increase of 8.36%. For MUV, GConv-Adapter (73.550%) outperforms full fine-tuning (72.288%) by 1.74%. Similarly, on HIV, GConv-Adapter (74.974%) shows a modest, but significant, improvement over full fine-tuning (72.695%) of 3.13%. These consistent improvements lead to a higher overall average ROC-AUC of 81.142% compared to full fine-tuning's 79.352%, representing a 2.25% improvement. When compared to other graph-specific PEFT methods, GConv-Adapter shows superior performance in most cases. Against G-Adapter, GConv-Adapter shows improvements on five out of six datasets: Tox21 (84.288% vs 83.343%), SIDER (82.641% vs 82.464%), ClinTox (97.558% vs 97.119%), MUV (73.550% vs 66.347%), and HIV (74.974% vs 69.103%). Only on BACE does G-Adapter perform marginally better (74.054% vs 74.002%). Compared to AdapterGNN, GConv-Adapter achieves better performance on four datasets: Tox21 (84.288% vs 83.977%), SIDER (82.641% vs 81.989%), ClinTox (97.558% vs 97.136%), and HIV (74.974% vs 72.982%). AdapterGNN shows slightly better performance on BACE and MUV. Overall, GConv-Adapter's average ROC-AUC of 81.142% surpasses both G-Adapter (78.738%) and AdapterGNN (80.489%) by 3.05% and 0.81% respectively. Compared to classical PEFT methods, GConv-Adapter demonstrates substantial improvements across all datasets. It outperforms the best classical PEFT method, Adapter (78.027%), by 3.99%, with even larger improvements over LoRA (76.903%), Surgical Fine-tuning (76.270%), and BitFit (68.042%).

We expect that if a careful hyperparameter search were performed for each dataset, GConv-Adapter could achieve even better results.

E Ablations

To assess the effectiveness of various architectural components, we conduct ablation studies whose results are summarized in Tables 9, 10, 11, and 12. The base configuration for the ablations is based on the definition given in equation 5, the position of the adapters is sequential (before and after the GNN layers, as later discussed in this section), for normalization we use BatchNorm for MPNNs and LayerNorm for GTs, and α is initialized to 1 for training.

Location First, we analyze the effect of inserting the GConv-Adapter at different stages within the GNN architecture: we systematically vary the positions where the adapter is inserted. Specifically, we explore three different insertion configurations: (1) before the GNN layers (pre), (2) after the GNN layers (post), and (3) both before and after the GNN layers (pre & post). Each configuration is evaluated in both sequential and parallel forms, where the former applies the adapter in a cascade with the original GNN, and the latter applies the adapter in parallel to the GNN operations. The results are presented in Table 9, comparing performance across both inductive and transductive learning tasks. The pre & post-sequential configuration consistently delivers the best overall performance, achieving the lowest RMSE on ESOL (1.165 ± 0.052) and Lipophilicity (0.720 ± 0.002), along with high ROC-AUC scores on SIDER (83.656 ± 0.550) and MUV (73.632 ± 1.902). In the transductive setting, it also achieves the highest accuracy on Cora (81.339 ± 1.290), Citeseer (75.310 ± 2.378), and PubMed (89.006 ± 0.187).

Table 9: Ablation Study on Adapter Insertion Forms for Inductive and Transductive Learning Tasks. The best results for each dataset are highlighted in **bold**.

(a) Inductive Learning Tasks

Positions	Type	ESOL (RMSE ↓)	Lipo (RMSE ↓)	SIDER (ROC-AUC ↑)	MUV (ROC-AUC ↑)
pre	sequential	1.178 ± 0.010	0.729 ± 0.018	83.552 ± 0.833	67.657 ± 0.386
	parallel	1.312 ± 0.032	0.920 ± 0.048	82.332 ± 0.463	65.312 ± 0.944
post	sequential	1.198 ± 0.012	0.727 ± 0.003	83.638 ± 0.737	69.950 ± 5.024
	parallel	1.202 ± 0.014	0.734 ± 0.004	82.860 ± 0.442	71.336 ± 1.875
pre & post	sequential	**1.165 ± 0.052**	**0.720 ± 0.002**	**83.656 ± 0.550**	**73.632 ± 1.902**
	parallel	1.246 ± 0.030	0.814 ± 0.064	81.065 ± 0.281	61.649 ± 1.805

(b) Transductive Learning Tasks

Positions	Type	Cora (Accuracy ↑)	Citeseer (Accuracy ↑)	PubMed (Accuracy ↑)
pre	sequential	78.533 ± 2.393	75.090 ± 2.097	88.695 ± 0.509
	parallel	73.880 ± 1.433	72.629 ± 2.150	87.762 ± 0.454
post	sequential	80.748 ± 1.267	74.910 ± 1.542	88.851 ± 0.412
	parallel	77.302 ± 0.664	73.629 ± 1.690	88.323 ± 0.269
pre & post	sequential	**81.339 ± 1.290**	**75.310 ± 2.378**	**89.006 ± 0.187**
	parallel	72.994 ± 1.663	73.069 ± 1.486	87.208 ± 0.282

Normalization To evaluate the impact of normalization layers on model performance, we conduct an ablation study by comparing two normalization strategies against a baseline with no normalization. In the inductive setting, we compare models with no normalization versus those using Batch Normalization; in the transductive setting, we compare no normalization with Layer Normalization. The results are presented in Table 10. For inductive tasks, Batch Normalization significantly improves performance by reducing the RMSE on FreeSolv from 2.346 ± 0.084 to 1.543 ± 0.260 and increasing the ROC-AUC on Tox21 from 85.380% to 85.781%. Surprisingly, in transductive tasks, the absence of normalization slightly outperforms LayerNorm. For the Cora dataset, no normalization achieves 80.847% accuracy compared to 80.601% with LayerNorm. Similarly, in the Citeseer dataset, no normalization reaches 75.510% accuracy versus 75.230% for LayerNorm. In the case of PubMed, both approaches yield identical results, 89.006%.

Learnable scalar We conduct an ablation study to examine the impact of a learnable scalar parameter by comparing models with and without it on inductive and transductive tasks. As shown in Table 11, enabling the learnable scalar improves performance for all datasets tested—for instance, in the inductive setting, the ESOL RMSE is reduced from 1.177 ± 0.023 to 1.156 ± 0.021 and the HIV ROC-AUC increases from 74.218 ± 0.926 to 74.441 ± 0.942, with similar improvements observed on the transductive benchmarks.

Skip connection Table 12 shows that incorporating skip connections enhances performance significantly across all datasets tested; for example, in the inductive setting, the FreeSolv RMSE decreases from 1.726 ± 0.230 to 1.543 ± 0.785 and the BACE ROC-AUC rises from $74.7643.307\%$ to $77.7843.032\%$, while in the transductive setting, accuracy on Cora improves dramatically from $29.7880.487\%$ to $80.6011.334\%$.

Table 10: Ablation study on normalization layers for inductive and transductive learning tasks. The best result for each dataset is highlighted in **bold**.

(a) Inductive Learning Tasks

Normalization	FreeSolv (RMSE ↓)	Lipo (RMSE ↓)	Tox21 (ROC-AUC ↑)	HIV (ROC-AUC ↑)
None	2.346 ± 0.084	0.720 ± 0.002	85.380 ± 0.470	72.852 ± 1.637
Batch Normalization	**1.543 ± 0.260**	**0.715 ± 0.016**	**85.781 ± 0.266**	**75.735 ± 1.014**

(b) Transductive Learning Tasks

Normalization	Cora (Accuracy ↑)	Citeseer (Accuracy ↑)	PubMed (Accuracy ↑)
None	**80.847 ± 1.159**	**75.510 ± 2.209**	**89.006 ± 0.187**
Layer Normalization	80.601 ± 1.489	75.230 ± 2.445	89.006 ± 0.187

Table 11: Ablation study on learnable scalar for inductive and transductive learning tasks. The best results for each dataset are highlighted in **bold**.

(a) Inductive Learning Tasks

Learnable Scalar	ESOL (RMSE ↓)	Lipo (RMSE ↓)	ClinTox (ROC-AUC ↑)	HIV (ROC-AUC ↑)
False	1.177 ± 0.023	0.726 ± 0.014	96.677 ± 0.225	74.218 ± 0.926
True	**1.156 ± 0.021**	**0.724 ± 0.008**	**96.730 ± 0.075**	**74.441 ± 0.942**

(b) Transductive Learning Tasks

Learnable Scalar	Cora (Accuracy ↑)	Citeseer (Accuracy ↑)	PubMed (Accuracy ↑)
False	80.551 ± 1.224	74.990 ± 2.678	89.006 ± 0.187
True	**80.798 ± 1.398**	**75.070 ± 2.114**	**89.033 ± 0.157**

Table 12: Ablation Study on Skip Connections for Inductive and Transductive Learning Tasks. The best results for each dataset are highlighted in **bold**.

(a) Inductive Learning Tasks

Skip Connection	FreeSolv (RMSE ↓)	Lipo (RMSE ↓)	BACE (ROC-AUC ↑)	MUV (ROC-AUC ↑)
False	1.726 ± 0.230	0.761 ± 0.029	74.764 ± 3.307	68.853 ± 4.823
True	**1.543 ± 0.785**	**0.723 ± 0.009**	**77.784 ± 3.032**	**71.445 ± 0.995**

(b) Transductive Learning Tasks

Skip Connection	Cora (Accuracy ↑)	Citeseer (Accuracy ↑)	PubMed (Accuracy ↑)
False	29.788 ± 0.487	22.529 ± 1.787	40.615 ± 0.427
True	**80.601 ± 1.334**	**74.950 ± 2.223**	**89.040 ± 0.154**