Do We Really Need Complicated Graph Learning Models? – A Simple but Effective Baseline

Kaan Sancak[*]
kaan@gatech.edu

Muhammed Fatih Balın[*]
balin@gatech.edu

Ümit V. Çatalyürek[† *]
umit@gatech.edu

Abstract

Despite advances in graph learning, increasingly complex models introduce significant overheads, including prolonged preprocessing and training times, excessive memory requirements, and numerous hyperparameters which often limit their scalability to large datasets. Consequently, evaluating model effectiveness in this rapidly growing field has become increasingly challenging. We investigate whether the complicated methods are necessary if foundational and scalable models can achieve better quality on large datasets. We first demonstrate that Graph Convolutional Network (GCN) is able to achieve competitive quality using skip connections on large datasets. Next, we argue that existing Graph Neural Network (GNN) skip connections are incomplete, lacking neighborhood embeddings within them. To address this, we introduce Neighbor Aware Skip Connections (NASC), a novel skip connection with an adaptive weighting strategy. Our evaluation show that GCN with NASC outperforms various baselines on large datasets, including GNNs and Graph Transformers (GTs), with negligible overheads, which we analyze both theoretically and empirically. We also demonstrate that NASC can be integrated into GTs, boosting performance across over 10 benchmark datasets with various properties and tasks. NASC empowers researchers to establish a robust baseline performance for large datasets, eliminating the need for extensive hyperparameter tuning, while supporting mini-batch training and seamless integration with popular graph learning libraries.

1 Introduction

Graph Neural Networks (GNNs) have become the de facto models for Graph Learning (GL) on node [1], link [2] and graph [3] level tasks, spanning diverse domains, such as computer vision, natural language processing, recommender systems, social network analysis, and material sciences [4]. GNNs utilize a message-passing scheme, where node embeddings are iteratively updated by propagating the embeddings of its neighbors followed by non-linear transformations. With l layers, the final representation of a node captures the structural information of its l-hop neighborhood.

Despite their success, GNNs suffer from multiple problems. Many foundational GNNs [1, 5, 6] achieve their best quality with 2/3-layer shallow networks. This is attributed to *over-smoothing* [7], which occurs when the vertex representations converge and eventually become indistinguishable as the model gets deeper. To overcome over-smoothing, various methods have incorporated skip connections [8, 9] to GNNs [1, 7, 10–13]. Secondly, GNNs face limitations in capturing *long-range dependencies* within graphs, where nodes need to exchange information over long distances [14]. This often leads to information *over-squashing* caused by repeated propagations [7, 15, 16]. Graph Transformers (GTs) address this limitation [17–20] by attending to potential neighbors among the entire set of nodes through self-attention [21]. However, GTs' quadratic complexity hinders their

[*]School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[†]Amazon Web Services. This publication describes work performed at the Georgia Institute of Technology and is not associated with AWS.

adoption for large datasets, limiting their application to small ones [14, 22–24]. Recent research has focused on addressing the scalability limitation through node tokenization [25, 26], dimensionality reduction [27], coarsening [28], and simplified attention mechanisms [29, 30].

The rapidly expanding field of GL necessitates effective baseline evaluation, yet this crucial task is becoming increasingly challenging due to several factors. First, many codebases only include dataloaders for specific datasets, requiring researchers to write new dataloaders and ensure data format consistency when adding new datasets. Second, many increasingly complex models have unfeasible time and memory requirements to large datasets, which is exacerbated when models are limited to full-batch training. Third, methods requiring numerous hyperparameters to be tuned for each dataset lead to a burdensome and time-consuming process, which becomes even more problematic for large datasets, especially when additional preprocessing is required. Compounding these issues, some codebases are not even available, and making it even more difficult to establish effective baselines.

In the current research landscape, we often overlook fundamental questions: Are we really making progress? Do we really need complicated methods if fundamental GNN models can achieve higher qualities while being scalable and widely available across GL frameworks? In this work, we first take a step back and provide our own implementation of GCN with Residuals (GCN + Res) [31], which combines skip connections [8] with Graph Convolution Network (GCN) [1]. We demonstrate that this fundamental baseline from 2019, a single line addition to GCN, can achieve competitive quality on large node prediction datasets, without additional hyper-parameters. Next, we observe that existing GNNs skip connection methods neglect neighborhood embeddings, focusing solely on individual vertex embeddings. Based on this insight, we propose Neighbor Aware Skip Connections (NASC), a novel approach that leverages neighborhood embeddings along with individual vertex embeddings in its skip connections with an adaptive weighting strategy. NASC can be applied to various GL models with minimal adjustments. To illustrate, we integrate NASC into GCN, and a recent GT, SGFormer [30], demonstrating consistent strong performance, often outperforming recent models across multiple benchmark datasets, encompassing heterophily and homophily graphs with node, graph, and link-level tasks. Furthermore, NASC requires only one additional hyper-parameter, which we found can be fixed at a specific value to consistently produce the best quality, and incurs in negligible overheads, which we discuss theoretically and validate empirically. Moreover, NASC is highly scalable and supports efficient graph mini-batch training.

The main motivation of this work is to provide a widely applicable, and easily implementable baseline that does not require extensive hyperparameter tuning, can be seamlessly integrated into any GL framework, and serves as a standard reference point for evaluating the effectiveness on large datasets.

2 Background and Related Work

A graph $G = (V, E)$ consist of vertices V and edges $E \subseteq V \times V$, where $n = |V|$ and $m = |E|$ denote the number of vertices and edges. $A \in \mathbb{R}^{n \times n}$ denotes the adjacency matrix of G, where a weighted edge $(u \to v) \subseteq E$ exists between source u and target v if $A_{u,v} \neq 0$. The feature embedding matrix $X^{(l)} \in \mathbb{R}^{n \times d^{(l)}}$ assigns a feature vector $x_v^{(l)} \in \mathbb{R}^{d^{(l)}}$ to each vertex v at layer l, with $X^{(0)}$ referring to input features. $N(v) = \{u \mid (u \to v) \in E\}$ denotes the incoming neighborhood set of v.

Graph Neural Networks (GNNs) can be represented under the framework of *propagations* (**P**) and *non-linear transformations* (**T**), [5, 6, 32]. In **P** step, a node v applies a propagation function, $\textsc{Prop}(\cdot)$, to the set of its neighborhood embeddings scaled by their edge weights to compute propagation embedding $h_v \in \mathbb{R}^d$ (layer notation omitted), as in Equation 1. In **T** step, v applies a non-linear function, $\textsc{Trans}(\cdot)$, to h_v to generate x_v, as in Equation 2. Under this framework, GCN [1] is expressed as below and more details including illustrations are provided in Appendix B.2.

$$h_v^{(l)} = \textsc{Prop}\big(\{\hat{A}_{u,v} x_u^{(l-1)}, u \in N(v) \cup \{v\}\}\big) = \sum_{u \in N(v) \cup \{v\}} \hat{A}_{u,v} x_u^{(l-1)} \tag{1}$$

$$x_v^{(l)} = \textsc{Trans}(h_v^{(l)}; W^{(l)}) = \sigma\big(h_v^{(l)} W^{(l)}\big) \tag{2}$$

where $\hat{A}$ is normalized A, with $W^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}}$, $d^{(l)}$ and $\sigma(\cdot)$ denoting layer weights, number of hidden units, and non-linear activation function. GCNs [1] also explicitly include self-loops in A.

Skip Connections. The original GCN [1] includes a residual version, similar to those in ResNet [8], which we refer to as traditional skip connection, see Equation 3. Skip connections enable input to bypass non-linear layers and propagate directly to deeper layers. This improves training dynamics by allowing gradients to flow directly between layers, addressing the vanishing gradients and improving robustness to input perturbations [8]. A skip connection is called residual [8] when using summation $(+)$, and dense [9] when using concatenation, [ ]. For more details, see Appendix B.1.

$$x_v^{(l)} = \textsc{Trans}(x_v^{(l-1)}, h_v^{(l)}; W^{(l)}) + x_v^{(l-1)} = \sigma\big(h_v^{(l)} W^{(l)}\big) + x_v^{(l-1)} \tag{3}$$

In GCN, a node equally contributes to $h_v^{(l)}$ as of its neighbors, see Equation 1. This can be problematic as a node's own features **could vanish** in deeper layers due to repeated pooling. GraphSAGE [5] tackles this by concatenating $h_v^{(l)}$ and $x_v^{(l-1)}$ before **T** and views it as a form of a skip connection:

$$h_v^{(l)} = \textsc{Prop}\big(\{\hat{A}_{u,v} x_u^{(l-1)}, u \in N(v)\}\big), \qquad x_v^{(l)} = \sigma\big([h_v^{(l)}, x_v^{(l-1)}] W^{(l)}\big) \tag{4}$$

Notice that this differs from traditional residuals that aim to improve training dynamics. In contrast, this skip connection arises from GNNs' inherent mechanism, enabling a node to treat its own embeddings distinctively from its neighbors, by training $x_v^{(l-1)}$ and $h_v^{(l)}$ with separate weights unlike Equation 3. We will refer these as topological skip connections.

Following up, DenseGCNs [31], ResGCN and DenseGCN, straightforwardly combine Equations 4 and 3. JK-Nets, a DenseNet variant, applies a final aggregation to all intermediate embeddings [10]. MixHop proposes another DenseNet variant with dense connections to a fixed number of consecutive layers, controlled by a hyper-parameter [33]. GCNII introduces residual connections from the initial input to every intermediate output, with additional hyperparameters controlling the initial residual and feature retention in subsequent layers [12]. For further details, please refer to Appendix B.2.

Categorization of GNNs. Most GNNs, inspired by GCN, can be categorized as *coupled models* [1, 5, 6, 10, 12, 31, 32], intertwine **P**s and **T**s, meaning that each **P** is strictly followed by a **T**. On the other hand, *decoupled models* [11, 34, 35, 35–38], first adopted by SGC [35], simplifies the training by first performing **P**s as a preprocessing step followed by a single non-linearity (**T**) at the end. Thi significantly reduces model complexity. However, its simplicity makes it less expressive and more prone to overfitting [35]. In contrast, other models [11, 39] perform **T**s followed by **P**s. While this simplifies the training, as the **P**s are performed after **T**s, they are still intertwined to some extent.

Graph Transformers (GTs) utilize Transformer architecture [21] within graphs. Its core component is multi-head self-attention (MHA) is a map from the input $X \in \mathbb{R}^{n \times d}$ to $\mathbb{R}^{n \times d}$ as:

$$Attn(X) = \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right), \qquad y = \mathrm{SelfAttention}(X) = Attn(H)V \tag{5}$$

here Q, K, V are linear projections of the input, and $Attn(H)$ captures pair-wise input similarities. As Equation 5 neglects graph topology in the adjacency matrix, and various structural and positional encodings have been proposed to incorporate graph topology into GTs [17, 18, 40–44]. Many of these methods, including the hybrid ones like GraphGPS [20], use Equation 5, which is problematic for large graphs due to quadratic time/space complexity $\mathcal{O}(n^2)$ and prolonged preprocessing times. Consequently, most GTs focus on small datasets like molecular ones [14, 22–24].

Recently, efforts have been made to simplify attention using linearized computations and dimensionalities [27, 29, 30, 39, 45, 46], while others focus on coarsening [28], sparsification [47]. Some approaches involve node tokenization [25, 26] to generate tokens as a preprocessing, and then apply attention on the node tokens, which can be seen as a decoupled model variation [48]. Despite numerous efforts on scalable GTs, the applicability them to large graphs remains questionable. Research has consistently shown that many proposed methods achieve worse quality than state-of-the-art GNNs [27, 29, 30]. In our work, we reveal that even a simple adjustment, as minimal as a single line, the earliest foundational GNN model can outperform many GTs designed for large datasets.

3 Methodology

3.1 Motivation

The field of GL has experienced exponential growth in recent years, with a surge in submissions to top machine learning conferences, where it has consistently ranked among the top 4 keywords, indicating the field's productivity. For supporting evidence, please refer to Appendix D. However, despite this growth, we question whether effective models for large datasets are being developed.

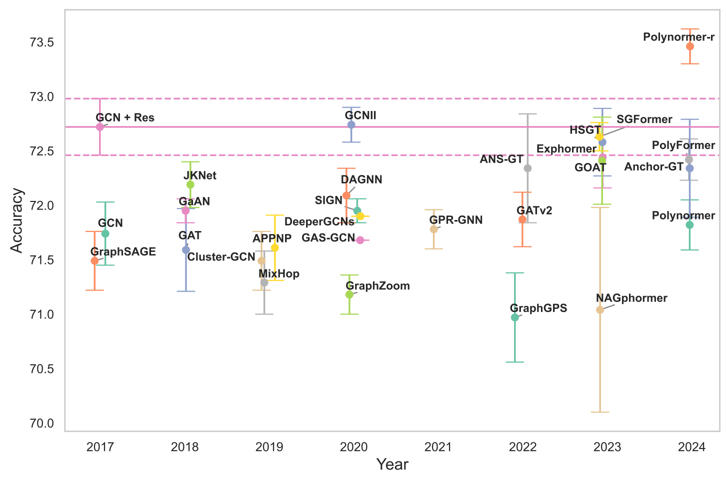

Figure 1: Test accuracy with standard deviation on ogbn-arxiv dataset across models over years.

To investigate, in Figure 1 we examine the test accuracy of various methods on the popular ogbn-arxiv benchmark over the years, with results gathered from the OGB leaderboard [3] and related work. To ensure a fairness, we exclude methods employing additional tricks such as additional features or correct and smooth [49]. For clarity, we focus on models achieving at least 70% accuracy, but a comprehensive figure can be found Appendix D. Additionally, we implemented our own GCN baseline with residuals (GCN + Res), as the OGB leaderboard's version includes additional tricks.

Observe that models proposed after 2021 are predominantly GTs or hybrid GNN & GT, whereas earlier models are mainly GNN-based. Notably, some of the earliest models outperform later ones. Specifically, GCN + Res, a simple baseline with better scalability than most recent models, no preprocessing and additional hyper-parameters compared to GCN, outperforms all baselines except one without extensive tuning. While the recent Polynormer's vanilla version underperforms GCN + Res, its Polynormer-r variant is the only model achieving higher quality, possibly due to different initialization and normalization strategies employed. Moreover, GCN + Res's ceiling (mean + std) is higher than all other baselines while maintaining a higher mean than majority. This raises the following research question: **(RQ1:)** Do we really need complicated models?

3.2 Challenges in Baseline Evaluation

Answering **RQ1** requires effective evaluation of the proposed models against comprehensive baselines and datasets. However, as the field rapidly expands, establishing effective baseline evaluations is becoming increasingly challenging as more models and datasets emerge. Surprisingly, recent works evaluating on large node prediction datasets [25, 27, 28, 30, 47, 50, 51] often lack simple yet effective baselines like GCN with residuals, which we demonstrated that can achieve higher test quality. This observation raises concerns about the comprehensiveness of existing baseline comparisons. Moreover, inconsistencies persist in the datasets and baselines included across different method evaluations, further complicating the ability to draw reliable conclusions about model effectiveness.

To better understand the root causes hindering robust evaluation, we identify several key contributing challenges: **(C1):** Codebases often only include dataloaders for specific datasets, requiring researchers to write new dataloaders and ensure data format consistency for new datasets. **(C2):** Increasingly complex models have unfeasible time and memory requirements. **(C3):** Methods requiring extensive hyperparameter tuning for each dataset lead to a burdensome and time-consuming process, especially for large datasets with additional preprocessing needs. **(C4):** Many models are limited to full-batch training, compounding the challenges C2 and C3. **(C5):** Some codebases are unavailable, further hindering effective evaluation. These challenges motivate **(RQ2:)** Can we establish accessible baselines for effective evaluation without burdening researchers with excessive coding and tuning?

[3]https://ogb.stanford.edu/docs/leader_nodeprop

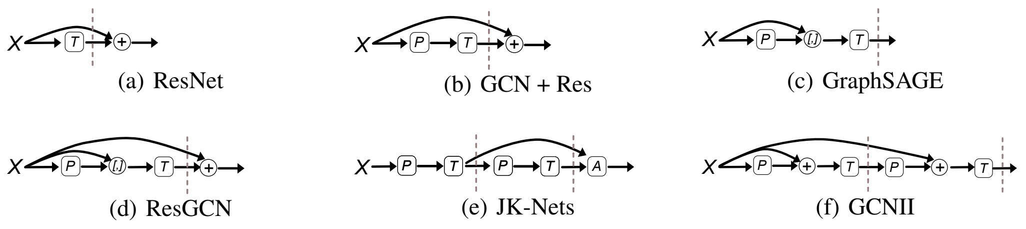

Figure 2: Model representations: (e), (f) 2 layers; others 1 layer. Dashed lines: layer boundaries.

3.3 An Effective, Scalable, Accessible yet Simple Baseline

To answer **RQ2**, we reconsider adaptation of skip connections in GNNs, which typically implement traditional skip connections, where the single vertex embedding x_v is added to the output, as shown in Fig. 2. However, input to the GNN transformation not limited to x_v, but also includes propagated l-hop embeddings, h_v, as shown in Eqs. 3 - 4. We argue h_v, which captures the iteratively aggregated neighborhood information can be leveraged as contextualized residuals to enhance model quality.

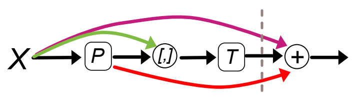

Figure 3: ResNet, GraphSAGE, NASC

Motivated by our insights, we introduce Neighbor Aware Skip Connection (NASC), incorporating propagation embeddings into GNN skip connections via *an adaptive weighting strategy* to adjust the residual terms. Including propagation residuals allows preserve and propagate contextual neighborhood information across layers. NASC can seamlessly integrate into various GL models, including GCN, GraphSAGE, SGFormer and others. For example, GraphSAGE with NASC is expressed as:

$$\begin{aligned} x_v^{(l)} &= \textsc{Trans}(h_v^{(l)}, x_v^{(l-1)}; W^{(l)}) + \alpha^{(l)} x_v^{(l-1)} + (1-\alpha^{(l)}) h_v^{(l)} \\ &= \sigma\big([h_v^{(l)}, x_v^{(l-1)}] W^{(l)}\big) + \alpha^{(l)} x_v^{(l-1)} + (1-\alpha^{(l)}) h_v^{(l)} \end{aligned} \tag{6}$$

In the context of GNN residual connections, NASC follows the same principles as traditional skip connections. Unlike GraphSAGE's topological skip connection, NASC allows propagation embeddings to directly flow to deeper layers without going under non-linear transformations. Different from decoupled models, in NASC transformations are still interviewed with neighborhood residuals.

Adaptive Weighting of Residuals. We express the residual term as a weighted sum controlled by α to avoid doubling the hidden dimension at every layer, and prevent increased memory requirements (**@C2**). We consider three forms of α: **(1)** Fixed scalar hyperparameter, or adaptively learned by:

$$\alpha^{(l)} = \sigma_{\text{sigmoid}}([x_v^{(l-1)}, h_v^{(l)}] W_\alpha^{(l)}) \tag{7}$$

where $W_\alpha^{(l)} \in \mathbb{R}^{2d^{(l-1)} \times q^{(l-1)}}$ is a learnable weight. **(2)** If $q^{(l-1)} = 1$, $\alpha^{(l)}$ is scalar scaling the matrices as a whole. **(3)** If $q^{(l-1)} = d^{(l-1)}$, it is a vector scaling the weights per hidden channel. The choice depends on the dataset and model complexity. Fixed scalars may be better for small datasets to avoid overfitting W_α, while learnable functions are better for larger and complex datasets.

Notably, NASC requires only one additional hyperparameter, α, compared to its backbone. Our empirical evaluation further shows that fixing $q = 1$ works best for large datasets. Thus, tuning NASC adds no extra hyperparameters or preprocessing to the backbone to practitioners (**@C3**).

Model Complexity and Overhead Analysis. In Appendix E, we break down single layer complexity of various models, including GCN, GraphSAGE, ResGCN, all exhibiting the same $\mathcal{O}(nd^2 + nm)$. To analyze computational overhead, we examine the number of FLOPS per layer. GCN requires $md + 2nd^2 + nd$ FLOPS for sparse matrix-matrix multiplication (SpMM), general matrix-matrix multiplication (GeMM), and activation. GraphSAGE adds $2nd^2$ FLOPS for concatenation, totaling $md + 4nd^2 + nd$. ResGCN adds nd FLOPS with an extra summation, totaling $md + 4nd^2 + 2nd$. NASC with fixed α, propagation residual summation adds nd FLOPS, totaling $md + 4nd^2 + 3nd$. With a learnable α, extra overheads include: GeMM ($2ndq$), softmax (nd), and element-wise multiplications ($2nd$), totaling $md + 4nd^2 + 4ndq + 5nd$ FLOPS. While $q = 1$, $7nd$ FLOPS overhead is negligible since $nd^2 \gg nd$, and if $q = d$, $4nd^2 + 3nd$ overhead is more considerable.

Importantly, as we implement NASC for mini-batch training (@**C4**), the n factor can be controlled, further diminishing the overhead with smaller batches. Section 4.3 verifies our theoretical analysis and demonstrates that NASC's overhead is negligible, especially with smaller batch sizes (@**C2,C4**).

Integration with Graph Transformers. Many state-of-the-art (SOTA) GT combine local and global modules including GraphGPS [20], Exphormer [47], Specformer [52], NodeFormer [29], SGFormer [30]. Local modules, such as GNNs, leverage the adjacency matrix, while global modules employ attention mechanisms, discarding the adjacency matrix. Notably, the skip connections of these models do not incorporate propagation residuals. For instance, SGFormer is expressed as:

$$z_v^{(l)} = (1-\theta)z_v^{(l)} + \theta \mathrm{GNN}(x_v^{(l-1)}, A), \qquad x_v^{(l)} = \mathrm{FFN}(z_v^{(l)}) \tag{8}$$

where z_v is the output of the global module, GNN is the local module, FFN is the final feed-forward layer, and θ is a hyper-parameter controlling the importance of local and global information. GNN can be an off-the-shelf GNN model, such as GCN (Eq. 3), making NASC a straightforward plug-in (@**C5**), similar to our illustration in Eq. 6. Consequently, we integrate NASC into SGFormer and conduct all the experiments from the original work in our evaluation.

Accessibility. To address the accessibility challenges in Section 3.2, we provide examples of NASC integration in PyTorch-Geometric (PyG) and Deep Graph Learning Library (DGL), illustrating how to plug it into various models with few lines of code (@**C5**). This enables NASC to work with any dataset within these frameworks, and we provide additional datasets, such as Pokec (@**C1**). Furthermore, NASC supports mini-batch training for large scale datasets (@**C2,C4**). Since NASC requires no additional hyper-parameter tuning (with $q = 1$) or preprocessing (@**C3**), it serves as a readily accessible, scalable, and effective baseline for researchers.

4 Evaluation

We benchmark NASC on diverse node-level tasks to assess its quality and scalability, including multi-task learning, long-range dependence, heterophily, and homophily, using a NVIDIA DGX Station 80GB XA100 4 GPU machine. Our evaluation consists of two main components: (1) Evaluation on standard large node level tasks with GNN backbone. (2) Evaluation on medium and large node level tasks with SGFormer backbone. We further analyze NASC's efficiency through ablation studies in Section 4.3, and also provide additional evaluations for other graph tasks in Section 4.4.

Implementation. We provide multiple implementations to cater to different scenarios, including a standalone full-batch training implementation using PyG, an integration with GraphBolt that enables graph mini-batch training with examples using PyG and DGL libraries, and integrations with the SGFormer and GraphGPS frameworks, offering flexibility and versatility for various use cases. We have included a portion of our code in the supplementary material, and the complete codebase will be made publicly available before or at the conclusion of the double-blind review process to ensure the reproducibility of our results.

4.1 Large Scale Node Level Graph Datasets with GNN Backbone

Table 1: Test accuracy on large-scale node prediction benchmarks: the **first**, **second**, and **third** best are highlighted. If available, we reuse the results from original papers [13, 37, 53, 54] and the OGB Leaderboard [55], otherwise we present our own findings.

Dataset	Flickr	Arxiv	Reddit	Yelp	Products
GCN	50.90 ± 0.12	71.74 ± 0.29	92.78 ± 0.11	40.08 ± 0.15	75.64 ± 0.21
SIGN	51.40 ± 0.10	71.95 ± 0.11	**96.80 ± 0.00**	63.10 ± 0.30	77.60 ± 0.13
GraphSAGE	**53.72 ± 0.16**	71.49 ± 0.27	**96.50 ± 0.03**	**63.03 ± 0.20**	78.70 ± 0.33
GIN	50.90 ± 0.12	71.74 ± 0.29	92.78 ± 0.11	40.08 ± 0.15	75.64 ± 0.21
GraphSaint	51.37 ± 0.21	67.95 ± 0.24	95.58 ± 0.07	29.42 ± 1.32	79.08 ± 0.24
ClusterGCN	48.10 ± 0.50	68.00 ± 0.59	95.40 ± 0.10	60.90 ± 0.50	78.97 ± 0.33
GAT	50.70 ± 0.32	71.59 ± 0.38	96.50 ± 0.14	61.58 ± 1.37	**79.45 ± 0.59**
ANG-GT	NA	72.34 ± 0.50	95.30 ± 0.81	NA	NA
HSGT	54.12 ± 0.51	72.58 ± 0.31	95.40 ± 0.10	63.47 ± 0.45	81.15 ± 0.13
GCN + Res	**53.40 ± 0.17**	**72.86 ± 0.16**	96.20 ± 0.03	**63.40 ± 0.70**	**79.30 ± 0.45**
GCN + NASC	**54.84 ± 0.39**	**73.24 ± 0.21**	**96.60 ± 0.03**	**64.38 ± 0.09**	**80.50 ± 0.16**

In Table 1, we first evaluate the efficacy of NASC on five large benchmark datasets commonly used to assess the performance of GNNs, with up to 2.5M nodes. Due to the large graph sizes, many recent GTs fail to run on these datasets. Therefore, our baselines for this comparison primarily include GNN baselines. However, we also include two GTs where the data loaders and results are available. For the arxiv and products datasets, we provide a more comprehensive table in the upcoming sections.

Although the vanilla GCN seems to underperform in some cases, simply adding residual connections (GCN + Res) achieves competitive results compared to more recent models, despite having lower computational complexity and being overlooked in other works' evaluations. Furthermore, the addition of neighborhood residuals (GCN + NASC) further improves the vanilla GCN's performance across all datasets, a simple modification without extensive hyperparameter tuning or preprocessing steps required by methods like ANS-GT, HSGT, or ClusterGCN. Notably, in some cases such as Reddit, where the dataset is highly saturated in terms of accuracy, recent works often report insignificant improvements with highly overlapping confidence intervals, and models such as GraphSAGE perform very competitively. While previous studies suggested complex models are necessary for better quality, our findings demonstrate that simple and scalable baselines, with proper components such as residual connections, can still achieve top performance, setting new thresholds for model evaluation. Our results highlight that algorithmic complexity is not always required. To the best of our knowledge, our score on the Flickr is currently one of the best reported accuracies with a vanilla GCN backbone, all while having insignificant overhead.

4.2 Medium and Large Scale Node Level Graph Datasets with GT Backbone

We integrate NASC into recent GT called SGFormer, and reproduce its experiments. We follow the exact setup including dataset splits used by SGFormer to produce consistent results.

Table 2: Test accuracy on medium-scale node prediction benchmarks: The **first**, **second**, and **third** best are highlighted.

Dataset	**Cora**	**CiteSeer**	**PubMed**	**Actor**	**Squirrel**	**Chameleon**	**Deezer**
# Nodes	2,708	3,327	19,717	7,600	2223	890	28,281
# Edges	5,278	4,552	44,324	29,926	46,998	8,854	92,752
Property	Homophilic	Homophilic	Homophilic	Heterophilic	Heterophilic	Heterophilic	Heterophilic
GCN	81.6 ± 0.4	71.6 ± 0.4	78.8 ± 0.6	30.1 ± 0.2	38.6 ± 1.8	41.3 ± 3.0	62.7 ± 0.7
GAT	83.0 ± 0.7	72.1 ± 1.1	79.0 ± 0.4	29.8 ± 0.6	35.6 ± 2.1	39.2 ± 3.1	61.7 ± 0.8
SGC	80.1 ± 0.2	71.9 ± 0.1	78.7 ± 0.1	27.0 ± 0.9	39.3 ± 2.3	39.0 ± 3.3	62.3 ± 0.4
JKNet	81.8 ± 0.5	70.7 ± 0.7	78.8 ± 0.7	30.8 ± 0.7	39.4 ± 1.6	39.4 ± 3.8	61.5 ± 0.4
APPNP	83.3 ± 0.5	71.8 ± 0.5	80.1 ± 0.2	31.3 ± 1.5	35.3 ± 1.9	38.4 ± 3.5	66.1 ± 0.6
H_2GCN	82.5 ± 0.8	71.4 ± 0.7	79.4 ± 0.4	34.4 ± 1.7	35.1 ± 1.2	38.1 ± 4.0	66.2 ± 0.8
SIGN	82.1 ± 0.3	72.4 ± 0.8	79.5 ± 0.5	36.5 ± 1.0	40.7 ± 2.5	41.7 ± 2.2	66.3 ± 0.3
CPGNN	80.8 ± 0.4	71.6 ± 0.4	78.5 ± 0.7	34.5 ± 0.7	38.9 ± 1.2	40.8 ± 2.0	65.8 ± 0.3
GloGNN	81.9 ± 0.4	72.1 ± 0.6	78.9 ± 0.4	36.4 ± 1.6	35.7 ± 1.3	40.2 ± 3.9	65.8 ± 0.8
Graphormer$_{\text{SMALL}}$	OOM	OOM	OOM	OOM	OOM	OOM	OOM
Graphormer$_{\text{SMALLER}}$	75.8 ± 1.1	65.6 ± 0.6	OOM	OOM	40.9 ± 2.5	41.9 ± 2.8	OOM
Graphormer$_{\text{ULTRASSMALL}}$	74.2 ± 0.9	63.6 ± 1.0	OOM	33.9 ± 1.4	39.9 ± 2.4	41.3 ± 2.8	OOM
GraphTrans$_{\text{SMALL}}$	80.7 ± 0.9	69.5 ± 0.7	OOM	32.6 ± 0.7	41.0 ± 2.8	42.8 ± 3.3	OOM
GraphTrans$_{\text{ULTRASSMALL}}$	81.7 ± 0.6	70.2 ± 0.8	77.4 ± 0.5	32.1 ± 0.8	40.6 ± 2.4	42.2 ± 2.9	OOM
NodeFormer	82.2 ± 0.9	72.5 ± 1.1	79.9 ± 1.0	36.9 ± 1.0	38.5 ± 1.5	34.7 ± 4.1	66.4 ± 0.7
SGFormer	$\mathbf{84.5 \pm 0.8}$	$\mathbf{72.6 \pm 0.2}$	$\mathbf{80.3 \pm 0.6}$	$\mathbf{37.9 \pm 1.1}$	$\mathbf{41.8 \pm 2.2}$	$\mathbf{44.9 \pm 3.9}$	$\mathbf{67.1 \pm 1.1}$
GCN + NASC	$\mathbf{83.7 \pm 0.9}$	$\mathbf{72.4 \pm 0.4}$	$\mathbf{80.1 \pm 0.6}$	$\mathbf{36.22 \pm 2.53}$	$\mathbf{43.7 \pm 1.5}$	$\mathbf{46.8 \pm 3.4}$	$\mathbf{72.8 \pm 4.9}$
SGFormer + NASC	$\mathbf{85.1 \pm 1.0}$	$\mathbf{73.5 \pm 0.3}$	$\mathbf{80.9 \pm 0.2}$	$\mathbf{39.82 \pm 1.40}$	$\mathbf{43.9 \pm 1.7}$	$\mathbf{50.8 \pm 5.6}$	$\mathbf{75.3 \pm 2.3}$

Discussion on Medium Scale Datasets. In Table 2, we first evaluate the efficacy of NASC on medium-scale datasets with number of nodes ranging from 2k-30k, compared to both GNN and GT baselines. Incorporating NASC in SGFormer consistently yields accuracy improvements across all datasets. Notably, on heterophilic datasets like Actor, Squirrel, Chameleon, and Deezer, the enhancements are more pronounced, with improvements up-to 8.1% on the largest graph, Deezer. This shows that integrating NASC into local component of GTs can lead to significant quality boosts.

Discussion on Large Scale Datasets. Next, in Table 3, we evaluate the efficacy of NASC on large-scale datasets with number of nodes ranging from 130K to 2.5M. Consistently, NASC enhances the performance of SGFormer with minimal modifications, requiring only a few lines of code, and without additional hyperparameters or preprocessing beyond the backbone model. Although improvements are modest in Amazon2m, with overlapping confidence intervals, NASC maintains quality even in the worst case. Moreover, across all other datasets, NASC consistently demonstrates improvements, with its lower bound (mean - std) surpassing the upper bound (mean + std) of other models.

Table 3: Test accuracy on large-scale node prediction benchmarks: the **first**, **second**, and **third** best are highlighted.

Method	**Proteins**	**Amazon2m**	**Pokec**	**Arxiv**
# nodes	132,534	2,449,029	1,632,803	169,343
# edges	39,561,252	61,859,140	30,622,564	1,166,243
Property	Multi-task	Long-range Depedence	Heterophilic	Homophilous
MLP	72.04 ± 0.48	63.46 ± 0.10	60.15 ± 0.03	55.50 ± 0.23
GCN	72.51 ± 0.35	83.90 ± 0.10	62.31 ± 1.13	71.74 ± 0.29
SGC	70.31 ± 0.23	81.21 ± 0.12	52.03 ± 0.84	67.79 ± 0.27
GCN-NSampler	73.51 ± 1.31	83.84 ± 0.42	63.75 ± 0.77	68.50 ± 0.23
GAT-NSampler	74.63 ± 1.24	85.17 ± 0.32	62.32 ± 0.65	67.63 ± 0.23
SIGN	71.24 ± 0.46	80.98 ± 0.31	68.01 ± 0.25	70.28 ± 0.25
NodeFormer	77.45 ± 1.15	87.85 ± 0.24	70.32 ± 0.45	59.90 ± 0.42
SGFormer	**79.53 ± 0.38**	**89.09 ± 0.10**	**73.76 ± 0.24**	**72.63 ± 0.13**
GCN + NASC	**80.53 ± 0.53**	**89.18 ± 0.07**	**75.02 ± 0.14**	**73.24 ± 0.21**
SGFormer + NASC	**80.46 ± 0.62**	**89.15 ± 0.17**	**74.74 ± 0.52**	**73.31 ± 0.10**

4.3 Ablation Study on Computational Overhead

Table 4: Overhead comparison of GCN + Res and GCN + NASC at batch sizes 1024 and 4096. Overheads are relative to the GCN + Res baseline, covering time/epoch, power, and peak memory.

Batch Size	**Model**	**q**	**Time/Epoch**	**Watt/it**	**Peak Mem. (MB)**
4096	GCN + Res	-	3.29	134.73	4011
	GCN + NASC	Fixed	$+3.34\%$	$+3.24\%$	$+2.34\%$
		1	$+6.99\%$	$+11.47\%$	$+17.51\%$
		256	$+7.90\%$	$+18.43\%$	$+24.52\%$
1024	GCN + Res	-	13.18	87.18	2903
	GCN + NASC	Fixed	-0.46%	$+0.03\%$	$+1.72\%$
		1	$+3.57\%$	$+0.20\%$	$+7.45\%$
		256	$+7.13\%$	$+2.18\%$	$+13.64\%$

In Table 4, we compare GraphSAGE with residual connections (ResGCN) and NASC on the ogbn-products dataset, one of the largest node-level benchmarks. We use a 3-layer model with a hidden size of 256 and sample 10 neighbors per layer with two batch sizes: 1024 and 4096. For NASC, we evaluate three variants: fixed scalar q, learnable q=1, and q=hidden dim. All models are trained for 1000 epochs on a cluster with four 80GB A100 GPUs. We discard the first 20% of epochs as warmup and report the mean and std of the remaining runs. Our analysis includes time per epoch, GPU power consumption per iteration, and peak GPU memory allocation during training.

The results indicate that the runtime overhead of using NASC with a fixed q is negligible. For a batch size of 4096, the overhead is only just 3.34%. Even when we scale the residuals as a whole (q = 1), the overhead is only 6.99%. In the worst case, when q = 256, the overhead reaches around 7.90%. Power consumption follows a similar trend. Peak memory usage of ResGCN and NASC with fixed q is nearly identical, while other versions incur 600-800MB (17.51% and 24.52%) overheads for a batch size of 4096; however, they still only allocate a very small portion of the available resources. Moreover, as NASC is implemented for minibatch training, reducing the batch size further minimizes the overhead. This is evident in the results for a batch size of 1024. GCN + Res and NASC with constant q yield nearly identical results across all metrics, while overheads for q=1 and q=256 decrease. Specifically, the memory overhead drops to 7.45% and 13.64%, with watt/it remaining within the same confidence intervals. Our findings confirm the analysis in Sec. 3.3, demonstrating that NASC does not incur significant overheads. Notably, our best results on large datasets are consistently achieved with q=1.

4.4 Further Evaluations

Graph Level Tasks. To demonstrate that NASC benefits other tasks beyond node classification, we extend our evaluation to graph-level tasks from the LRGB benchmark [14]: peptides-struct

Table 5: Evaluation on Graph Level Tasks [14].

Method	Pep.-Func (AP $\uparrow$)	Pep.-Struct (MAE$\downarrow$)
Transformer	0.6326 ± 0.0126	0.2529 ± 0.0016
SAN	0.6439 ± 0.0075	0.2545 ± 0.0012
GINE	0.5498 ± 0.0079	0.3547 ± 0.0045
+ Tuning	0.6621 ± 0.0067	0.2473 ± 0.0017
GatedGCN	0.6069 ± 0.0035	0.3357 ± 0.0006
+ Tuning	0.6765 ± 0.0047	0.2477 ± 0.0009
GPS	0.6535 ± 0.0041	0.2500 ± 0.0005
+ Tuning	0.6534 ± 0.0091	0.2509 ± 0.0014
GCN	0.5930 ± 0.0023	0.3496 ± 0.0013
+ Tuning	0.6860 ± 0.0050	0.2460 ± 0.0007
+ NASC	**0.7079 ± 0.0029**	**0.2440 ± 0.0007**

Table 6: Evaluation on PPI.

Model	Micro-F1
GraphSAGE	61.20
GAT	97.30
JKNet	97.60
GeniePath	98.50
GaAN	98.71
Cluster-GCN	99.36
SAGE + Res (ResGCN)	98.25
SAGE + NASC	99.39

(graph regression) and peptides-func (graph classification), which have been utilized in various works [20, 47, 56]. Our implementation is based on a recent benchmark [5] that showed simple GNNs can achieve competitive performance compared to complex ones like GTs.

Table 5 indicates that graph-level tasks can indeed benefit from neighbor residuals, with a significant improvement observed on peptides-func. Notably, our model employs a simple GCN backbone, in contrast to computationally intensive baselines like GraphGPS that use dense attention mechanisms. This graph-level evaluation complements our main focus on node-level tasks, illustrating the versatility of NASC across different graph learning paradigms.

Inductive Setting. In Table 6, we further extend our evaluations to the inductive setting. We report the micro-averaged F1 scores on the nodes of the two test graphs. For ResGCN and NASC, we average over 10 runs using the parameters with the highest validation accuracy. For others, we reuse the results reported in [5, 6, 10, 12, 54, 57, 58]. Our study demonstrates that incorporating propagation embeddings in the residuals improves the performance of deep GNNs in the inductive setting as well. NASC outperforms its counterpart ResGCN, specifically improving performance by 0.85%. This improvement highlights the effectiveness of NASC in generalizing to unseen graphs, a critical aspect of many real-world applications.

Table 7: Comparison of NASC and NASC$^{\oplus}$ on PPI and Arxiv.

Method	PPI	Arxiv
NASC	99.39 ± 0.01	73.24 ± 0.21
NASC$^{\oplus}$	99.44 ± 0.01	73.18 ± 0.22

Residual vs Dense NASC. In Table 7, we implement a dense version of NASC, called NASC$^{\oplus}$ and compare the two architectures. We have observed that NASC and NASC$^{\oplus}$ achieve similar results for both Arxiv and PPI datasets. With similar performance, one might question which model is a better choice. To answer this question, we have examined the number of parameters used by each model and found that NASC$^{\oplus}$ can achieve similar results to NASC using 25% fewer parameters, despite both configurations using the same number of layers and the width of NASC$^{\oplus}$ increases linearly. This is because NASC$^{\oplus}$ is able to match NASC using a smaller number of hidden units.

5 Conclusion

In this work, we introduced Neighbor Aware Skip Connections (NASC), which utilizes propagation embeddings in its skip connections. Through experimental evaluations on multiple benchmark datasets, we showed that NASC consistently outperforms traditional GNN residual methods and various baselines, particularly on larger datasets where training takes longer. Despite its seemingly modest nature, our approach has surprisingly been overlooked in the literature. If the proposed method consistently improves performance without any negative effects, then it should be preferred over existing methods. The future direction includes adding support for other GNN backbones.

Acknowledgements

We would like to thank Jay Wang and Elif Ceren Fitoz for their support and feedback. This work was partially supported by the NSF grant CCF-1919021.

References

[1] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 1, 2, 3, 14

[2] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018. 1

[3] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 1

[4] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020. 1

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 1, 2, 3, 9, 17

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 1, 2, 3, 9

[7] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018. 1

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 3, 14

[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1, 3, 14, 15

[10] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018. 1, 3, 9, 15

[11] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 338–348, 2020. 3

[12] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020. 3, 9, 15

[13] Wentao Zhang, Zeang Sheng, Ziqi Yin, Yuezihan Jiang, Yikuan Xia, Jun Gao, Zhi Yang, and Bin Cui. Model degradation hinders deep graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2493–2503, 2022. 1, 6

[14] Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022. 1, 2, 3, 8, 9

[15] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2020. 1

[16] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=7UmjRGzp-A`. 1

[17] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020. 1, 3

[18] 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 A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=OeWooOxFwDa`. 3

[19] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021.

[20] Ladislav Rampasek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=lMMaNf6oxKM`. 1, 3, 6, 9

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 3

[22] Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. A large-scale database for graph representation learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL `https://openreview.net/forum?id=1xDTDk3XPW`. 2, 3

[23] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[24] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023. 2, 3

[25] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. NAGphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=8KYeilT3Ow`. 2, 3, 4

[26] Dongqi Fu, Zhigang Hua, Yan Xie, Jin Fang, Si Zhang, Kaan Sancak, Hao Wu, Andrey Malevich, Jingrui He, and Bo Long. VCR-graphormer: A mini-batch graph transformer via virtual connections. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=SUUrkC3STJ`. 2, 3

[27] Kezhi Kong, Jiuhai Chen, John Kirchenbauer, Renkun Ni, C. Bayan Bruss, and Tom Goldstein. GOAT: A global transformer on large-scale graphs. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 2023. 2, 3, 4

[28] Wenhao Zhu, Tianyu Wen, Guojie Song, Xiaojun Ma, and Liang Wang. Hierarchical transformer for scalable graph learning. *arXiv preprint arXiv:2305.02866*, 2023. 2, 3, 4

[29] Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=sMezXGG5So`. 2, 3, 6

[30] Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. Sgformer: Simplifying and empowering transformers for large-graph representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2, 3, 4, 6, 17

[31] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019. 2, 3, 15

[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 2, 3

[33] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019. 3

[34] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2464–2473, 2020. 3

[35] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019. 3

[36] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.

[37] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael M. Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *CoRR*, 2020. 6

[38] Wentao Zhang, Zeang Sheng, Mingyu Yang, Yang Li, Yu Shen, Zhi Yang, and Bin Cui. Nafs: A simple yet tough-to-beat baseline for graph representation learning. In *International Conference on Machine Learning*, pages 26467–26483. PMLR, 2022. 3

[39] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. 3

[40] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=huAdB-Tj4yG`. 3

[41] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

[42] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022.

[43] Haiteng Zhao, Shuming Ma, Dongdong Zhang, Zhi-Hong Deng, and Furu Wei. Are more layers beneficial to graph transformers? In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=uagC-X9XMi8`.

[44] Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, Puneet K Dokania, Mark Coates, Philip Torr, and Ser-Nam Lim. Graph inductive biases in transformers without message passing. In *International Conference on Machine Learning*, pages 23321–23337. PMLR, 2023. 3

[45] Krzysztof Choromanski, Han Lin, Haoxian Chen, Tianyi Zhang, Arijit Sehanobish, Valerii Likhosherstov, Jack Parker-Holder, Tamas Sarlos, Adrian Weller, and Thomas Weingarten. From block-toeplitz matrices to differential equations on graphs: towards a general theory for scalable masked transformers. In *International Conference on Machine Learning*, pages 3962–3983. PMLR, 2022. 3

[46] Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. Difformer: Scalable (graph) transformers induced by energy constrained diffusion. In *The Eleventh International Conference on Learning Representations*, 2022. 3

[47] Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, 2023. 3, 4, 6, 9

[48] Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *International Conference on Learning Representations*, 2023. 3

[49] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=8E1-f3VhX1o`. 4

[50] Wenhao Zhu, Guojie Song, Liang Wang, and Shaoguo Liu. Anchorgt: Efficient and flexible attention architecture for scalable graph transformers. *arXiv preprint arXiv:2405.03481*, 2024. 4

[51] Chenhui Deng, Zichao Yue, and Zhiru Zhang. Polynormer: Polynomial-expressive graph transformer in linear time. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=hmv1LpNfXa`. 4

[52] Deyu Bo, Chuan Shi, Lele Wang, and Renjie Liao. Specformer: Spectral graph neural networks meet transformers. In *The Eleventh International Conference on Learning Representations*, 2022. 6

[53] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJe8pkHFwS`. 6, 17

[54] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019. 6, 9

[55] 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. *Advances in neural information processing systems*, 33:22118–22133, 2020. 6, 16

[56] Xiaoxin He, Bryan Hooi, Thomas Laurent, Adam Perold, Yann LeCun, and Xavier Bresson. A generalization of vit/mlp-mixer to graphs. In *International Conference on Machine Learning*, pages 12724–12745. PMLR, 2023. 9

[57] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. Geniepath: Graph neural networks with adaptive receptive paths. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4424–4431, 2019. 9

[58] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018. 9

[59] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015. 14

[60] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 2011. 14

[61] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016. 14

[62] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 14

[63] Tianjun Yao, Jiaqi Sun, Defu Cao, Kun Zhang, and Guangyi Chen. Mugsi: Distilling gnns with multi-granularity structural information for graph classification. In *Proceedings of the ACM on Web Conference 2024*, pages 709–720, 2024. 15

[64] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016. 15

[65] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. 16

[66] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft Academic Graph: When experts are not enough. *Quantitative Science Studies*, 1(1), 2020. 16

[67] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 16

[68] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017. 17

[69] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 17

A Appendix

B Background Details

B.1 Skip Connections in Neural Networks

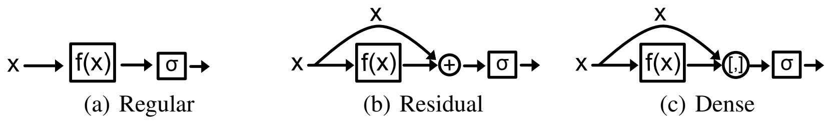

Figure 4: Representation of different neural network blocks: (a) a regular block, (b) a residual block where the input is added to the output, and (c) a dense block where each intermediate output is concatenated, denoted by [ ]. The function f(x) represents the composite function applied to the input.

Given an input $X^{(0)}$, a feed-forward convolutional neural network (CNN) [59] consisting of L layers applies a non-linear transformation $f^{(l)}(\cdot)$ to $X^{(l)}$ to produce output $X^{(l+1)}$ at each layer l, $l \in \{1, 2, ..., L\}$. The function $f^{(l)}$ is parameterized by a weight matrix $W^{(l)}$. This transformation typically includes a composite function $f^{(l)}(\cdot)$ such as linear, convolution, pooling, normalization, dropout, and other layers (see Figure 4(a)). Ignoring biases for simplicity, the feed-forward process can be represented by:

$$X^{(l)} = f^{(l)}(X^{(l-1)}; W^{(l)}) \tag{9}$$

allow input to bypass one or multiple layers, and directly propagate to a deeper layer. In the case of ResNets, the skip connection, also called residual connection, is an identity mapping [8]. The input to the layer is added to the output of the layer, bypassing the non-linear transformation via identity mapping, as shown in Figure 4(b). Hence, we obtain:

$$X^{(l)} = f^{(l)}(X^{(l-1)}; W^{(l)}) + X^{(l-1)} \tag{10}$$

Residual connections improve training dynamics by allowing gradients to flow directly between layers, addressing the vanishing gradient problem and improving robustness to input perturbations [8].

Densely Connected Networks (DenseNets) [9] utilize concatenation, denoted by [ ], instead of summation, as in Figure 4(c). DenseNet layers receive inputs from all the previous layers, allowing features to be reused in subsequent layers. DenseNets are expressed as:

$$X^{(l)} = [f^{(l)}(X^{(l-1)}; W^{(l)}), X^{(l-1)}] \tag{11}$$

B.2 Graph Neural Networks (GNNs)

Graph Convolutional Networks (GCNs).

Motivated by spectral graph convolutions [60, 61], Graph Convolutional Networks (GCNs) [1] are a specific form of Message Passing Neural Networks (MPNNs) [62]. GCNs use standard neural network optimization techniques which are generally faster and more scalable than traditional spectral methods based on eigen-decomposition. The l-th layer of a GCN can be expressed as in Equations 1 and 2, where $\hat{A}$ is the normalized adjacency matrix, which usually comes in two forms: $\hat{A} = \frac{1}{\sqrt{D^-}} A \frac{1}{\sqrt{D^+}}$ or $\hat{A} = \frac{1}{D^-} A$, but other normalization can be integrated as well.

The propagation (**P**) and transformation (**T**) framework of GNNs can be can be interpreted at both the graph and node levels. In Section2, we provide a node level interpretation, which is also illustrated in the bottom part of Figure 5. At a graph level, a sparse matrix-matrix multiplication (SpMM) between $\hat{A}$ and $X^{(l-1)}$ computes **P**, which is followed by **T**, as illustrated in the top part of Figure 5.

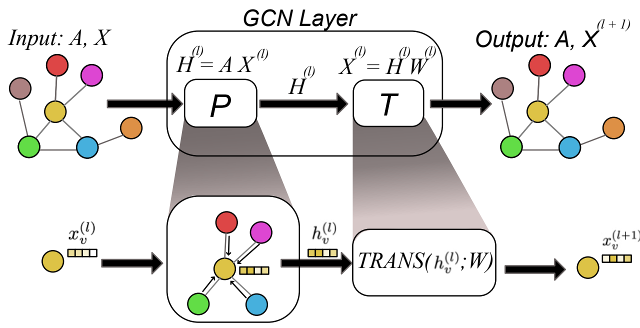

Figure 5: GNN representations: Graph (top) and node (bottom) levels

DeepGCNs. Inspired by ResNets, ResGCN [31] is expressed as:

$$\begin{aligned} h_v^{(l)} &= \textsc{Prop}\big(\{\hat{A}_{u,v} x_u^{(l-1)}, u \in N(v)\}\big) \\ x_v^{(l)} &= \textsc{Trans}(h_v^{(l)}, x_v^{(l-1)}; W^{(l)}) + x_v^{(l-1)} \\ &= \sigma\big([h_v^{(l)}, x_v^{(l-1)}] W^{(l)}\big) + x_v^{(l-1)} \end{aligned} \tag{12}$$

ResGCN combines the skip connection schemes from Equation 4 and Equation 3 by allowing intermediate representations to have a path to both identity mapping and transformation through concatenation. Similarly, inspired by DenseNets [9], DenseGCN [31] uses concatenation over summation enabling reuse of intermediate representations in subsequent layers.

$$\begin{aligned} h_v^{(l)} &= \textsc{Prop}\big(\{\hat{A}_{u,v} x_u^{(l-1)}, u \in N(v)\}\big) \\ x_v^{(l)} &= \big[\textsc{Trans}(h_v^{(l)}, x_v^{(l-1)}; W^{(l)}), x_v^{(l-1)}\big] \\ &= \big[\sigma\big([h_v^{(l)}, x_v^{(l-1)}] W^{(l)}\big), x_v^{(l-1)}\big] \end{aligned} \tag{13}$$

JK-Nets, a variant of DenseNet, applies a final aggregation, referred to as PROP-FINAL, to all the intermediate embeddings, regardless of the specific PROP and TRANS functions used [10]. The PROP-FINAL function can be a concatenation, max-pooling, or LSTM-attention.

$$x_v^{(L+1)} = \textsc{Prop-Final}\big(\{x_v^{(1)}, x_v^{(2)}, \dots, x_v^{(L)}\}\big) \tag{14}$$

GCNII [12] is expressed as:

$$\begin{aligned} h_v^{(l)} &= \text{P-GCN}\big(\{\hat{A}_{u,v} x_u^{(l-1)}, u \in N(v) \cup \{v\}\}\big) \\ x_v^{(l)} &= \textsc{Trans}(h_v^{(l)}, x_v^{(l-1)}; W^{(l)}) \\ &= \sigma\big((1-\alpha^{(l)}) h_v^{(l)} + \alpha^{(l)} x_v^{(0)})((1-\beta^{(l)})I + \beta^{(l)} W^{(l)})\big) \end{aligned} \tag{15}$$

Here, $\alpha^{(l)}$ is a hyperparameter that regulates the initial residual and controls the proportion of input features retained in subsequent layers. β_l follow a decay function $\beta_l = \frac{\lambda}{l}$ where λ is a hyperparameter. While the authors claim that I establishes an identity mapping similar to the ones in ResNet, we argue that this is not exact case, as the residual is applied to the weights before the activation, unlike in ResNet. Furthermore, we note that $\beta^{(l)}$ helps with the initialization of $W^{(l)}$ by setting the diagonal of $W^{(l)}$ to $(1-\beta^{(l)})$. This decaying $\beta^{(l)}$ ensures that later layers are initialized close to I, which can be beneficial on very small datasets where overfitting occurs quickly. In these scenarios, the model will not perturb the initial features from earlier layers in the later layers.

Some other recent related work includes MuGSI [63] which introduces a framework for distilling knowledge from GNNs to MLPs specifically for graph classification tasks. This approach allows for efficient structural knowledge distillation at different levels of granularity, including graph-level, subgraph-level, and node-level distillation.

Table 8: Properties of benchmark datasets.

DATASET	n	m	$\frac{m}{n}$	$d^{(0)}$	$d^{(L)}$
CORA	2.7K	5.4K	2.03	1.4K	7
CITESEER	3.3K	4.7K	1.42	3.7K	6
PUBMED	20K	44K	2.25	500	3
PHOTO	7.5K	119K	15.90	745	8
COMPUTERS	13K	246K	18.36	767	10
PPI	57K	820K	14.38	50	121
FLICKR	90K	900K	10.08	500	7
ARXIV	170K	1.2M	6.89	128	40
REDDIT	233K	115M	492	602	41
YELP	717K	14M	19.5	300	100
PRODUCTS	2.5M	62M	25.26	100	47

C Experimental Details

C.1 Platenoid Datasets

Details This is a collection of three datasets (Cora, CiteSeer, PubMed) that are commonly used for GNN evaluation. The task is to classify the subjects of the documents. The nodes and edges represent academic papers and citations respectively. The feature vectors are bag of words representations of the respective papers. We apply the standard training, validation, and test split as outlined in [64] for the datasets under examination. The split consists of 20 nodes per class for training, 500 nodes for validation, and 1000 nodes for testing.

Hyperparameter Tuning. We conducted a comprehensive hyperparameter search optimizing the accuracy on validation nodes over 1000 trials for each method, including the learning rate, dropout rate, number of hidden units, and weight decay per layer. For NASC, we drop $\alpha^{(l)}$ and compute and compute $x_v^{(l)} = \textsc{Trans}(h_v^{(l)}, x_v^{(l-1)}; W^{(l)}) + x_v^{(l-1)} + h_v^{(l)}$ due to the small volume of the data.

C.2 Amazon

This is a collection of two datasets from Amazon co-purchase network, where nodes represent products and edges are present between two nodes if they are frequently bought together, and the task is to classify the categories [65]. The node features are bag of word representations of the user reviews. To the best of our knowledge, there is no standard fix splits for Amazon datasets. Therefore, following the traditional practice in the community, we use random splits with 20 training and 30 validation nodes per class, and the rest is used for testing. In specific, for each evaluation and tuning run, we use different random seeds to have a fair comparison.

Hyperparameter Tuning. We conduct a comprehensive hyperparameter search optimizing the accuracy on validation nodes over 1000 trials for GCN, GCNII, NASC/NASC$^{\oplus}$, and ResGCN/DenseGCN, including the number of layers, learning rate, dropout rate, weight decay and number of hidden units. For GCNII, we also tune α, λ, and additional weight decay for the linear layers at the beginning and end of the network. For each tuning trial, we use 20 random splits. Similar to Platenoid datasets, for NASC, we drop $\alpha^{(l)}$ and compute $x_v^{(l)} = \textsc{Trans}(h_v^{(l)}, x_v^{(l-1)}; W^{(l)}) + x_v^{(l-1)} + h_v^{(l)}$.

C.3 Arxiv and Products

Arxiv is a citation network from Open Graph Benchmark (OGB) [55] node prediction datasets that consists of CS arxiv papers by Microsoft Academic Graph (MAG) [66]. Similarly, the nodes and edges represent academic papers and citations respectively. The feature vectors are bag of words representations of the respective papers. We apply the standard 54%-18%-28% split for training, validation and testing. For NASC, we simultaneously learn vector $\alpha^{(l)}$ with the model training. The Products dataset, on the other hand, is a co-purchase network extracted from Amazon. Features are bag-of-words representations of the product descriptions, and they are reduced to 100 dimensions using Principal Component Analysis Nodes represent various products, and edges signify products that are frequently bought together. The task is to identify the product category.

Hyperparameter Tuning. We conduct a comprehensive hyperparameter search optimizing the accuracy on validation nodes over 1000 trials for GCN, NASC/NASC$^{\oplus}$, and ResGCN/DenseGCN, including the number of layers, learning rate, dropout rate and number of hidden units. Our best performing method, NASC is trained on 6 layers with 716 hidden units using Adam optimizer with 0.00136 learning rate and no weight decay and a dropout rate of 0.48. We also apply batch normalization [67].

C.4 Flickr and Yelp

The Flickr and Yelp datasets are acquired from their corresponding networks [53]. In the Flickr dataset, nodes represent uploaded images, and connections occur between nodes with shared properties. The Yelp dataset connects nodes that are considered friends within the social network. We use publicly available standard split for both of the datasets. We follow the same hyperparameter tuning procedure as Arxiv.

C.5 Reddit

The Reddit dataset [5] is sourced from Reddit posts, where each node represents a post, and connections exist between posts commented on by the same user. The task involves classifying posts into their respective subreddits (communities). We use publicly available standard split. We follow the same hyperparameter tuning procedure as Arxiv.

C.6 PPI

The protein-protein interaction (PPI) dataset consists of graphs representing human tissues [68]. Following the previous work [5], we use 20 graphs for training, 2 for validation and 2 for testing. For NASC, we simultaneously learn vector $\alpha^{(l)}$ with the model training.

Hyperparameter Tuning. We conduct a comprehensive hyperparameter search optimizing the accuracy on validation nodes over 1000 trials for GCN, NASC and NASC$^{\oplus}$ including the number of layers, learning rate, dropout rate and number of hidden units. Our best performing method, NASC$^{\oplus}$ is trained on 6 layers with 2048 hidden units using Adam optimizer with 0.01 learning rate and no weight decay. We also apply a dropout rate of 0.4. For GCNII, we use the hyperparameters suggested by the authors and utilize the open-source code available at PyTorch-Geometric (PyG) library [69].

C.7 Other Datasets

For other datasets, we follow the exact setup in SGFormer [30].

D Motivation Details

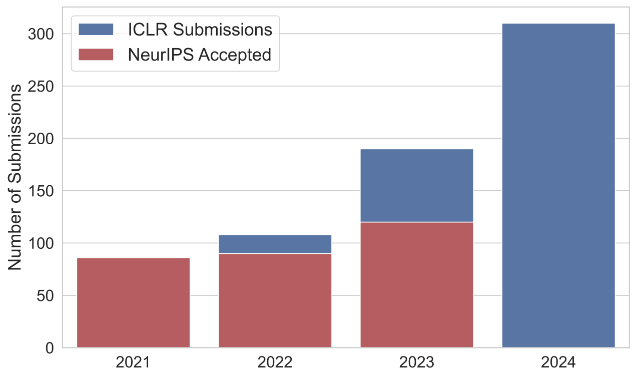

Figure 6: ICLR submitted and NeurIPS accepted submissions over years

E Complexity Analysis

In this analysis, we evaluate the computation overhead of NASC compared to traditional counterparts for different choices of $\alpha^{(l)}$. To begin, let's break down Equations 1 and 2 into its three components: **T**, **P**, and σ. The **T** step involves a Generalized Matrix Matrix Multiplication (GeMM) between $X^{(l)} \in \mathbb{R}^{n\times d}$ and $W^{(l)} \in \mathbb{R}^{d\times d}$, resulting in a complexity of $\mathcal{O}(nd^2)$. The **P** step can be calculated using a GeMM between $A \in \mathbb{R}^{n\times n}$ and the output of **T**, $Z^{(l+1)} \in \mathbb{R}^{n\times d}$, resulting in a complexity of $\mathcal{O}(n^2 d)$. However, since real-world graphs are usually sparse ($m \ll n^2$), we can utilize Sparse Matrix Matrix Multiplication (SpMM) to compute the **P** step, reducing the complexity to $\mathcal{O}(md)$. The element-wise operation σ on $X^{(l+1)} \in \mathbb{R}^{n\times d}$ introduces a complexity of $\mathcal{O}(nd)$. Therefore, the overall complexity of GCN is $\mathcal{O}(nd^2 + md + nd) = \mathcal{O}(nd^2 + md)$. GraphSAGE introduces concatenation but maintains the same complexity as GCN. It changes the **T** complexity to $\mathcal{O}(2nd^2) = \mathcal{O}(nd^2)$, while the overall complexity remains unchanged. ResGCN incorporates vertex embeddings in its residual, requiring an additional pass over $X^{(l)}$. However, this doesn't affect the complexity, and it remains the same as GraphSAGE. For NASC, when α is determined by a fixed scalar, the only additional computation needed is the summation of $X^{(l)}$ and $AX^{(l-1)} \in \mathbb{R}^{n\times d}$. The overall complexity remains the same: $\mathcal{O}(nd^2 + md + nd + nd) = \mathcal{O}(nd^2 + md)$. When $\alpha^{(l)}$ is determined by a vector, an additional GeMM is performed between $W_\alpha^{(l)}$ and $[X^{(l)}, AX^{(l-1)}]$, followed by a softmax operation. The additional GeMM yields $\mathcal{O}(2ndq) = \mathcal{O}(ndq)$, and the softmax operation yields $\mathcal{O}(nq)$. Once $\alpha^{(l)}$ is calculated, an element-wise multiplication is performed between $\alpha^{(l)}$ and the residual terms as shown in Equation 6, resulting in a complexity of $\mathcal{O}(nd + nd) = \mathcal{O}(nd)$. Additionally, the summation requires extra passes over the residual terms. The overall complexity of NASC is $\mathcal{O}(nd^2 + nm + nd + nq + nd) = \mathcal{O}(nd^2 + nm + nq + nd)$. Notably, for both choices of $q = 1$ or $q = d$, the complexity of NASC reduces to $\mathcal{O}(nd^2 + nm)$.

F Additional Experiments

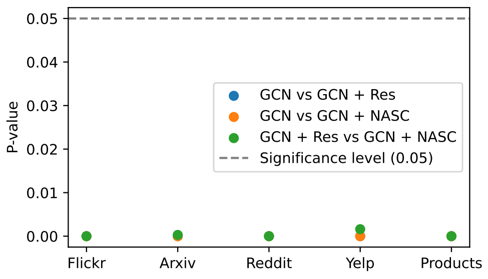

Figure 7: P-values Across Datasets for Each Model Comparison for Table 1

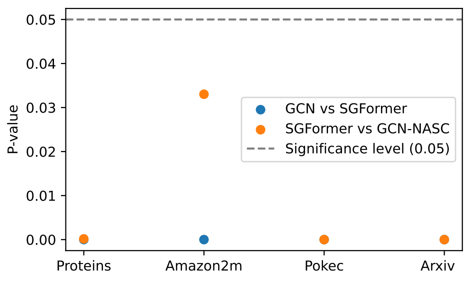

Figure 8: P-values Across Datasets for Each Model Comparison for Table 3

Table 9: P-values for Pairwise Comparisons of Model Performance

Dataset	GCN vs GCN + Res	GCN vs GCN + NASC	GCN + Res vs GCN + NASC
Flickr	9.58e-12	2.96e-17	1.36e-07
Arxiv	3.50e-10	4.01e-08	2.90e-04
Reddit	5.00e-17	1.57e-16	8.95e-17
Yelp	9.42e-32	3.10e-16	1.61e-03
Products	7.79e-21	7.90e-12	6.09e-06

When we look at the t-statistics using Anova, pairwise model accuracy comparisons for each dataset indicates that the performance are significantly different between compared models. For Table 9, we see that NASC significantly improves GCN and GCN + RES. Moreover, when we look at Table 10, SGFormer improves upon GCN, and GCN + NASC significantly outperforms SGFormer across all datasets.

Table 10: P-values for Pairwise Comparisons of Model Performance

Dataset	GCN vs GCN-NASC	GCN vs SGFormer	GCN-NASC vs SGFormer
Proteins	3.95e-17	1.70e-19	1.68e-04
Amazon2m	3.96e-26	2.49e-27	3.30e-02
Pokec	3.33e-11	3.65e-11	5.84e-10
Arxiv	3.50e-10	9.82e-07	1.15e-06