Scalable Expressiveness through Preprocessed Graph Perturbations

Danial Saber
danial.saber@ontariotechu.ca
Ontario Tech University
Oshawa, Ontario, Canada

Amirali Salehi-Abari
abari@ontariotechu.ca
Ontario Tech University
Oshawa, Ontario, Canada

ABSTRACT

Graph Neural Networks (GNNs) have emerged as the predominant method for analyzing graph-structured data. However, canonical GNNs have limited expressive power and generalization capability, thus triggering the development of more expressive yet computationally intensive methods. One such approach is to create a series of perturbed versions of input graphs and then repeatedly conduct multiple message-passing operations on all variations during training. Despite their expressive power, this approach does not scale well on larger graphs. To address this scalability issue, we introduce *Scalable Expressiveness through Preprocessed Graph Perturbation (SE2P)*. This model offers a flexible, configurable balance between scalability and generalizability with four distinct configuration classes. Our extensive experiments demonstrate that SE2P can enhance generalizability compared to benchmarks while achieving significant speed improvements of up to 8-fold.[1]

KEYWORDS

Graph Neural Networks, Scalability, Graph Perturbation

ACM Reference Format:
Danial Saber and Amirali Salehi-Abari. 2024. Scalable Expressiveness through Preprocessed Graph Perturbations. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3627673.3679993

1 INTRODUCTION

Graph Neural Networks (GNNs) have applications in various domains, such as recommender systems [48], protein modeling [16], educational systems [37], and knowledge graph completion [1]. However, graph data's complexity, scale, and dynamic nature pose substantial challenges to GNNs, emphasizing the importance of improving their generalization and computational efficiency.

Message-passing GNNs (MPNNs), a popular class of GNNs, facilitate the exchange of messages between nodes to integrate their local structural and feature information within a graph. However, MPNNs are limited by the 1-dimensional Weisfeiler-Lehman (1-WL) graph-isomorphism test [27], and are not scalable to large graphs.

Several approaches have been proposed to enhance the computational efficiency of MPNNs, such as removing intermediate non-linearities in GNN layers [13, 14, 32, 47, 59], graph downsampling during preprocessing [10, 31, 44, 55] or sampling during message-passing [8, 9, 18, 21, 51, 62]. All these approaches are still limited by the 1-WL expressivity constraint. To go beyond 1-WL expressive power, a wide variety of solutions have been proposed, suffering from scalability issues: *Higher-order GNNs* [29, 33] require (at least) cubic computational complexity for message passing [57, 60]; *Feature-augmented GNNs* require computing computationally-expensive features like structural encodings [3, 6], geodesic distances [28, 46, 58], and positional encodings [12]; and *Subgraph GNNs* [4, 22, 23, 39–41, 53] typically involve extracting multiple overlapping large subgraphs, which their cumulative size are significantly large, sometimes to hundreds of times the size of the original graph, rendering them impractical for large graphs.

Our approach. We introduce *Scalable Expressiveness through Preprocessed Graph Perturbation (SE2P)*, a model combining flexibility, scalability, and expressiveness. Our approach offers four configuration classes, each offering a unique balance between scalability and generalizability. Through preprocessing, SE2P generates multiple perturbations of the input graph by a perturbation policy (e.g., random node removal) and diffuses nodal features across each perturbed graph. Despite its diffusion similarity to SGCN [47] and its variants [13, 14, 32, 59], SE2P leverages the expressive power offered by multiple perturbed graphs [4, 39] to surpass 1-WL expressiveness limits. The flexibility of SE2P is of practical importance, allowing for the selection of learnable or non-learnable aggregation functions and thus enabling scalable or expressive variations of many models. Our empirical results demonstrate significant speedup (up to 8×) and enhanced generalizability for SE2P compared to baselines.

2 PREDICTION TASK AND BACKGROUND

We consider an undirected graph $G = (V, E)$ with $|V| = n$ nodes, $|E| = m$ edges, and adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. Each node $i \in V$ possesses the d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^n$, which can be viewed as the i-th row of $n \times d$ feature matrix $\mathbf{X}$.

Prediction Task. Graph classification or regression involves predicting a label (e.g., carcinogenicity classification [45]) or a property (e.g., molecule solubility level [17]) for an entire graph based on its structure and associated features (e.g., node or edge features). Specifically, the task is formulated as a supervised learning problem, aiming to learn a mapping function $f : \mathcal{G} \rightarrow \mathcal{Y}$, given a labeled dataset $D = \{(G_i, y_i)\}$, where $\mathcal{G}$ is input space, and $\mathcal{Y}$ is class label space (or real for regression), G_i is input graph sample, and y_i is an

[1] An extended version of this work is available in [42].

CIKM '24, October 21–25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679993

expected label (or property). Although GNNs have demonstrated significant success in graph classification or regression tasks [19], their expressive power is limited by the 1-WL test [36, 49].

Perturbed GNNs. To overcome 1-WL expressivity limitation, *Perturbed GNNs* (e.g., DropGNN [39]) applies a shared GNN on R different perturbations of the input graph (during both training and testing). For each perturbation $(\mathbf{A}_r, \mathbf{X}_r)$, some graph structure (e.g., nodes or edges) is randomly changed. For example, DropGNN randomly drops out some nodes for each perturbation. In perturbed GNNs, a shared L-layer GNN operates on each perturbation to generate perturbed node representations $\mathbf{Z}_r = \text{GNN}(\mathbf{A}_r, \mathbf{X}_r)$. These perturbed embeddings are then merged into final node embeddings using an aggregator function: $\mathbf{Z} = \text{MERGE}(\mathbf{Z}_1, \cdots, \mathbf{Z}_R)$. Through multiple perturbations, the model observes slightly perturbed variants of the same L-hop neighborhood around any node. Thus, even if the non-isomorphic neighborhoods are indistinguishable by the standard GNNs, their randomly modified variants are more likely distinguishable, yielding higher expressive power. However, Perturbed GNNs (e.g., DropGNN) face major scalability issues as the number of perturbations increases in large datasets.

Simplified Diffusion-Based Models. An approach to enhance the scalability of GNNs is simplifying their architectures by eliminating their intermediate non-linearities [13, 14, 32, 47, 59]. This technique allows for the precomputation of feature propagation and further acceleration. For instance, SGCN [47] removes intermediate non-linearities in an L-layer GCN, predicting node class labels $\mathbf{Y}$ using $\mathbf{Y} = \sigma(\mathbf{A}^L\mathbf{X}\mathbf{W})$, where σ is a non-linear function, and $\mathbf{W}$ is a weight matrix. The diffusion term $\mathbf{A}^L\mathbf{X}$ can be precomputed before training. SIGN [14] extends SGCN by considering a set of diffusion matrices rather than just one. The diffusion terms in our model share some similarities with SIGN. However, unlike our method, the expressivity of SGCN and SIGN is bounded by the 1-WL.

3 SE2P

Inspired by the expressive power of methods relying on generating perturbations (e.g., DropGNN [39]), we propose ***S**calable **E**xpressiveness through **P**reprocessed Graph **P**erturbations (SE2P)*. In SE2P, we first generate different perturbations of the input graph (e.g., through random node dropout) to improve expressiveness. The scalability is offered by once precomputing feature diffusions over perturbed graphs and removing the need for message-passing during training. As illustrated in Fig. 1, SE2P generates a set of R graph perturbations $\{(\mathbf{A}_r, \mathbf{X}_r)\}$ for graph G with adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$. Although SE2P accommodates any perturbation kind (e.g., node removal, subgraph sampling, etc.), we here consider random nodal removal as a perturbation due to its theoretical expressiveness power [39]. In each perturbation $(\mathbf{A}_r, \mathbf{X}_r)$, any node of the original graph G is removed with probability p. Each perturbed adjacency matrix $\mathbf{A}_r$ is normalized by $\hat{\mathbf{A}}_r = \mathbf{D}_r^{-\frac{1}{2}}\mathbf{A}_r\mathbf{D}_r^{-\frac{1}{2}}$, where $\mathbf{D}_r$ is the diagonal matrix of $\mathbf{A}_r$. To emulate the message-passing of GNNs on perturbed graphs, we apply feature diffusion by $\hat{\mathbf{A}}_r\mathbf{X_r}$. Similarly, the message passing of an L-layer GNN can be emulated by $\hat{\mathbf{A}}_r^L\mathbf{X_r}$, which can be once precomputed before the training for each perturbed graph as a preprocessing step. To enhance node representation in each perturbed graph, we emulate jumping knowledge [49] by

$$\mathbf{Z}_r = \text{COMBINE}(\mathbf{X}, \hat{\mathbf{A}}_r^1\mathbf{X}_r, \cdots, \hat{\mathbf{A}}_r^L\mathbf{X}_r), \tag{1}$$

where COMBINE combines all the virtual L layer's output with the original feature matrix into the node embedding matrix of the perturbed graph. The examples of COMBINE can be simple readout-type operators (e.g., column-wise vector concatenation) or learnable adaptive aggregation mechanisms (e.g., DeepSet [54]). When the simple non-learnable operator is deployed, we compute $\mathbf{Z}_r$ through preprocessing steps for more speedup.

The next step is to aggregate node representations of perturbed graphs $\{\mathbf{Z}_r\}$ to a single nodal representation matrix $\mathbf{Z}$:

$$\mathbf{Z} = \text{MERGE}(\mathbf{Z}_1, \cdots, \mathbf{Z}_R), \tag{2}$$

where several options exist for MERGE ranging from non-learnable aggregation methods (e.g., element-wise mean) to learnable set aggregations (e.g., DeepSet). While non-learnable aggregation methods such as averaging provide simplicity and computational efficiency, they risk overriding and blending information across perturbed graphs, possibly leading to the loss or dilution of discriminative information. However, all computations up to this point can occur during the preprocessing phase, provided that aggregations in Eqs. 1 and 2 are non-learnable. This preprocessing offers a considerable speedup since the message-passing of a multi-layer GNN on multiple perturbed graphs is emulated by one-time preprocessing steps rather than iterative computations during training. When more expressiveness is desired over scalability, one can employ learnable aggregation over perturbed graphs.

For graph prediction tasks, we then apply a POOL function to aggregate nodes' final representations into a graph representation $\mathbf{z}_G = \text{POOL}(\mathbf{Z})$, where POOL function can be non-learnable (e.g., element-wise sum) or a learnable graph pooling method. Non-learnable functions can speed up computation, specifically if they are precomputable. However, they reduce the model's expressiveness by lacking non-linearities. If higher expressiveness is desired, given some computational budget, one might consider learnable graph pooling methods such as hierarchical or top-k pooling [7, 15, 25, 52], global soft attention layer [30], set-transformer [26], or even MLP combined non-learnable aggregators (e.g., sum or mean). After pooling, the graph representation $\mathbf{z}_G$ undergoes learnable non-linearities to get the class probabilities.

How does SE2P trade-off scalability and expressivity? The SE2P's aggregation functions COMBINE, MERGE, and POOL balance scalability and expressivity, configurable as either learnable or non-learnable. This creates four practical *configuration classes* within SE2P, where each class is identified by which aggregator is learnable or not. Configuration C1 maximizes scalability by making all functions non-learnable, allowing maximal precomputation before training. Configuration C2 improves expressivity with a learnable POOL, while COMBINE and MERGE remain non-learnable and pre-computable. Configuration C3 further enhances expressivity by making POOL and MERGE learnable. Configuration C4, with all learnable functions, offers the highest expressivity but the least scalability due to minimal preprocessing. Moving from C1 to C4 increases expressiveness but reduces scalability, as it allows less preprocessing to ease training's computational burden.

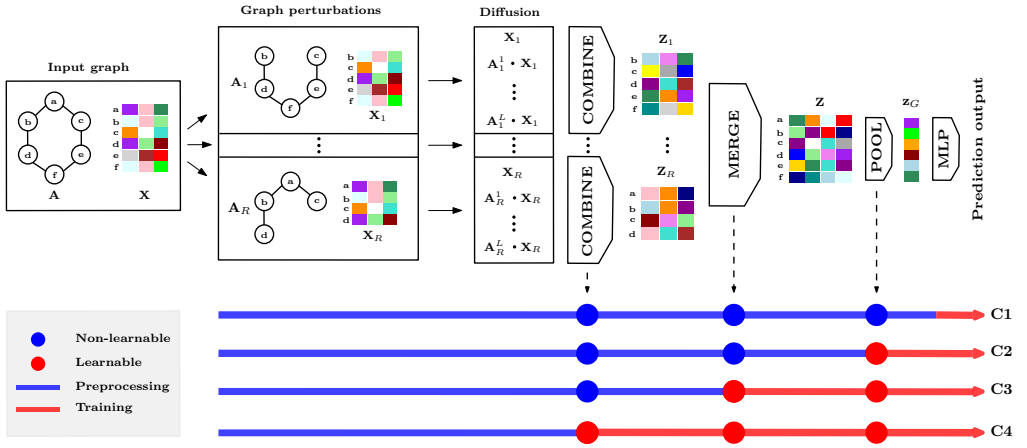

Figure 1: *SE2P* first generates R perturbations of the input graph with new adjacency and feature matrices $(\mathbf{A}_r, \mathbf{X}_r)$. Next, node features are diffused for each perturbation by a set of diffusion matrices. Then, the COMBINE function combines these diffused features for each perturbed graph into feature matrices $\mathbf{Z}_r$. All these matrices then undergo the MERGE function to generate the graph's representation matrix $\mathbf{Z}$. POOL is then applied to create a graph representation z_G, which is transformed by an MLP to the predicted output. The functions COMBINE, MERGE, and POOL are either non-learnable (blue circle) or learnable (red circle). This flexibility allows us to choose between different configuration classes (C1–C4) to balance scalability, achieved by including more preprocessing steps (blue line), and expressivity, achieved by more learnability (red line).

Table 1: Preprocessing and inference time complexities. R is the number of perturbations, L is the number of (virtual) layers, n is the number of nodes, m is the number of edges, and d is the feature and hidden dimensions.

	DropGNN	SE2P-C1	SE2P-C2	SE2P-C3	SE2P-C4
Prep.	$O(1)$	$O(RLmd)$	$O(RLmd)$	$O(RLmd)$	$O(RLmd)$
Inf.	$O(LR(nd^2+md))$	$O(d^2)$	$O(nd^2)$	$O(Rnd^2)$	$O(RLnd^2)$

Our configurations for SE2P. We implemented and studied four instances of SE2P, covering all configuration classes, with specific COMBINE, MERGE, and POOL functions. SE2P-C1 (maximum scalability) uses column-wise vector concatenation for COMBINE, element-wise mean for MERGE, and element-wise sum pooling for POOL. SE2P-C2 replaces the sum pooling of SE2P-C1 with a learnable POOL function, which consists of an MLP followed by element-wise sum pooling. SE2P-C3 is the same as SE2P-C2 except for leveraging Deepsets as a learnable MERGE function. The least scalable but most expressive configuration is SE2P-C4, which replaces the non-learnable COMBINE of SE2P-C3 to Deepsets. Table 1 summarizes our running time analyses of these variants [42].

4 EXPERIMENTS

Our experiments aim to empirically validate the scalability and generalizability of our SE2P models against various benchmarks.

Datasets. We experiment with four datasets from the TU datasets collection [35] (PROTEINS [5, 11], PTC-MR [45], IMDB-M [50], and COLLAB [50]) and two datasets from Open Graph Benchmark [20] (OGBG-MOLHIV and OGBG-MOLTOX). For TU datasets, We use the same dataset splitting deployed by other studies [4, 39, 49, 61] whereas for OGB we use their provided scaffold splits.

Baselines. For TU datasets, we compare our model against WL subtree [43], DCNN [2], DGCNN [56], PATCHY-SAN [38], IGN [34], GCN [24], GIN [49], DropGIN [39], and DropGCN.[2] On OGB datasets, we compare against GCN, GIN, DropGCN, and DropGIN.

Table 2: Average validation accuracy (%), TU datasets. The best result is in bold. In parenthesis: the ranks of our model against baselines (1st, 2nd, and 3rd are colored), and comparison to DropGNNs (●=better, ●=comparable with difference < 0.2, and ●=worse). OOM denotes out of memory.

Model	PROTEINS	PTC-MR	IMDB-M	COLLAB
WL subtree	75.0 ± 3.1	59.9 ± 4.3	50.9 ± 3.8	78.9 ± 1.9
DCNN	61.3 ± 1.6	56.6 ± 1.2	33.5 ± 1.4	52.1 ± 2.7
DGCNN	75.5 ± 0.9	58.6 ± 2.5	47.8 ± 0.9	73.7 ± 0.4
PATCHYSAN	75.0 ± 2.5	62.3 ± 5.7	45.2 ± 2.8	72.6 ± 2.2
IGN	76.6 ± 5.5	58.5 ± 6.9	48.7 ± 3.4	78.3 ± 2.5
GIN	75.4 ± 5.0	63.9 ± 8.3	51.5 ± 4.0	82.2 ± 2.1
GCN	75.9 ± 5.5	64.2 ± 9.7	52.0 ± 4.1	82.6 ± 2.2
DropGIN	76.1 ± 5.1	65.2 ± 9.8	52.3 ± 3.8	OOM
DropGCN	76.1 ± 5.8	64.5 ± 9.1	52.1 ± 3.3	OOM
SE2P-C1	74.7 ± 5.7 (9, ●)	64.5 ± 8.0 (2, ●)	52.1 ± 2.8 (2, ●)	79.8 ± 1.8 (3, ●)
SE2P-C2	77.6 ± 6.3 (1, ●)	65.1 ± 7.3 (2, ●)	52.3 ± 2.3 (1, ●)	83.3 ± 2.1 (1, ●)
SE2P-C3	**77.6 ± 5.0** (1, ●)	**66.2 ± 6.8** (1, ●)	**52.9 ± 3.5** (1, ●)	**83.5 ± 1.7** (1, ●)
SE2P-C4	76.8 ± 4.7 (1, ●)	66.1 ± 8.8 (1, ●)	52.4 ± 2.4 (1, ●)	82.8 ± 2.1 (1, ●)

Experimental setup. To fairly compare DropGNN [39] and SE2P variants, we used DropGNN's recommended hyperparameters: dropping node probability $p = \frac{2}{1+\gamma}$ and number of perturbations $R = \lfloor \gamma \rfloor$, where γ is the dataset's average node degree. We set the number of (virtual) layers to $L = 2$ or 3. For TU benchmark evaluations, we present the accuracies of the WL subtree kernel, DCNN, DGCNN, PATCHY-SAN, and IGN, as reported in their original papers, which all share the same experimental setup as ours, adopted from [49]. Under this experimental setup, we also reproduced the results for GCN [24], GIN [49], DropGIN [39], and DropGCN for our scalability comparisons. We grid-searched the hyperparameters for these baselines and SE2P variants on the recommended search spaces

[2] Our introduced DropGCN has replaced GIN layers with GCN layers in DropGIN.

Table 3: Runtimes on TU datasets. Inference time (Inf.) is the time per epoch (avg. over 350 epochs). Run includes preprocessing time (Pre.) and total inference time. SE2P are color-coded by faster, comparable, and slower than any of baselines. The speedup corresponds to the ratio of time taken by the slowest baseline compared to our model. Pre. and inf. are in seconds, while Run is in minutes.

	PROTEINS			PTC-MR			IMDB-M			COLLAB		
Model	**Pre.**	**Inf.**	**Run**	**Pre.**	**Inf.**	**Run**	**Pre.**	**Inf.**	**Run**	**Pre.**	**Inf.**	**Run**
GIN	–	0.74	4.3	–	1.52	8.8	–	0.72	4.2	–	3.87	22.5
GCN	–	0.71	4.1	–	0.64	3.7	–	0.73	4.2	–	2.62	15.3
DropGIN	–	0.86	5.0	–	1.74	10.1	–	0.94	5.4	–	OOM	OOM
DropGCN	–	0.94	5.4	–	0.86	5.0	–	1.19	6.9	–	OOM	OOM
SE2P-C1	8.7	0.27	1.7	1.8	0.22	1.3	4.0	0.23	1.4	230.3	0.26	5.3
Speedup	–	3.48	3.19	–	7.90	7.70	–	5.17	4.97	–	14.88	4.21
SE2P-C2	8.7	0.41	2.5	1.7	0.28	1.6	4.0	0.37	2.2	224.2	0.72	7.9
Speedup	–	2.29	2.16	–	6.21	6.02	–	3.13	3.14	–	5.37	2.84
SE2P-C3	8.5	1.34	7.9	1.6	0.70	4.1	3.1	0.82	4.8	220.2	20.06	120.6
Speedup	–	0.70	0.68	–	2.48	2.46	–	1.45	1.43	–	0.19	0.18
SE2P-C4	8.5	7.06	41.3	1.4	3.58	20.9	3.0	2.61	15.3	214.7	42.86	253.5
Speedup	–	0.13	0.13	–	0.48	0.48	–	0.45	0.45	–	0.09	0.08

Table 4: Average ROC-AUC (%) over 10 runs, OGB datasets. The best is in Bold. The preprocessing and the inference time are in seconds. The total runtime is in minutes. Color-coding is faster, comparable, and slower than any of baselines.

Model	**OGBG-MOLHIV**				**OGBG-MOLTOX21**			
	Test	**Prep.**	**Inf.**	**Run**	**Test**	**Prep.**	**Inf.**	**Run**
GIN	74.0±1.9	–	3.5	5.9	72.7±1.7	–	1.6	2.6
GCN	74.1±1.9	–	3.5	5.9	72.2±1.1	–	1.7	2.8
DropGIN	OOM	–	–	–	73.6±1.0	–	2.3	3.8
DropGCN	OOM	–	–	–	72.1±1.2	–	2.5	4.2
SE2P-C1	71.4±1.3	170.5	1.8	5.8	71.6±0.5	22.6	0.6	1.3
SE2P-C2	74.0±1.4	169.7	2.2	6.5	72.9±0.6	22.5	0.8	1.8
SE2P-C3	**74.5 ± 2.6**	171.6	11.2	21.5	73.5±1.0	22.6	2.4	4.4
SE2P-C4	OOM	173.1	–	–	**74.1 ± 1.0**	22.6	11.3	19.3

[39]. For the OGB benchmark, we employed the same hyperparameter tuning of the TU benchmark, and then followed the evaluation procedure proposed in [20]: we ran each experiment with 10 different random seeds, and models were optimized using Adam for 100 epochs. We report the average test accuracies corresponding to the best average validation accuracy.[3]

Results and Discussions. Table 2 shows the validation accuracy results on TU datasets. SE2P-C3 outperforms other SE2P configurations and baselines across all datasets, improving generalizability over all baselines ranging from 0.6% (in IMDB-M) to 1.5% (in PROTEINS). SE2P-C2 and SE2P-C4 also show competitive performance, securing the top three-ranked methods among all baselines for all datasets. For instance, SE2P-C4 improves or shows comparable results to all baselines in all datasets. Our least expressive SE2P-C1 model performs sub-optimally on the PROTEINS dataset but is relatively competitive in other datasets (e.g., PTC-MR, IMDB-M, COLLAB) by being ranked among the top three of baselines. The poor performance in PROTEINS might be due to the lack of non-linearity before obtaining the graph representation and complexity of the dataset. Except SE2P-C1, models with perturbations outperform baselines without graph perturbation (e.g., GCN, IGN), indicating that graph perturbations are a simple yet effective method for enhancing generalization. Compared to DropGNN with graph perturbations, SE2P configurations (except SE2P-C1) show comparable or better generalizability and handle scalability issues, avoiding OOM in COLLAB and reducing training times for other datasets.[4]

We further compare the runtime efficiency of SE2P configurations with GCN, GIN, DropGCN, and DropGIN in Table 3. SE2P's preprocessing time ranges from almost 2 seconds for PTC-MR to 4 minutes for COLLAB. Across all datasets, SE2P-C1 and SE2P-C2 are faster than the baselines in training and total runtime (including preprocessing and training time over all epochs). The speedup for total runtime ranges from 3.19× (in PROTEINS) to 7.70× (in PTC-MR) for SE2P-C1, and from 2.16× (in PROTEINS) to 6.02× (in PTC-MR) for SE2P-C2. SE2P-C3 has comparable runtime to the baselines (except for COLLAB) while improving generalizability. SE2P-C4 is the slowest model due to its longer training time. Overall, if 3-6× scalability with comparable generalizability is desired, SE2P-C2 is the best option. For maintaining baseline scalability while consistently improving generalizability, SE2P-C3 is recommended.

Table 4 shows the results on the OGB datasets. In OGBG-MOLHIV, SE2P-C2 achieves comparable results to the baselines while offering a speedup of roughly 30%. SE2P-C3 outperforms baselines but at the cost of longer training times. DropGCN, DropGIN, and SE2P-C4 faced out-of-memory issues, primarily due to the large number of graphs (48, 127 graphs) and extensive message-passing over many graph perturbations (for DropGCN and DropGIN) and feature transformation over diffusion sets of each perturbation (for SE2P-C4). In OGBG-MOLTOX, all methods utilizing node-dropout perturbations (except SE2P-C1, which lacks sufficient non-linearity and expressivity) outperform the two baselines without graph perturbations. For comparable performance and faster runtime, SE2P-C2 is preferred. It demonstrates roughly a 30% speed improvement over the fastest baseline (GIN) and a 130% speed improvement over the slowest baseline (DropGCN). For higher generalization, SE2P-C3 and SE2P-C4 are recommended despite reduced scalability.[5]

5 CONCLUSION AND FUTURE WORK

We introduced SE2P, a flexible framework with four configuration classes that balance scalability and generalizability. SE2P leverages graph perturbations and feature diffusion in the preprocessing stage and offers choices between learnable and non-learnable aggregators to achieve the desirable scalability-expressiveness balance. Our experiments on an extensive set of benchmarks validate the effectiveness of SE2P. Future directions include exploring other graph perturbation policies, providing theoretical analyses of graph perturbations through the lens of matrix perturbation theory, and developing adaptive methods for selecting the number of perturbations.

[3]The code is available at https://github.com/Danial-sb/SE2P.

[4]We encountered out-of-memory issues for DropGNN on the COLLAB dataset due to the large number of perturbations per graph. Reducing the batch size allows these models to run, but the results were suboptimal. We report OOM to highlight computational bottlenecks rather than expressiveness concerns.

[5]Our sensitivity analyses in the extended version [42], show that, in most cases, SE2P models with sub-optimal hyperparameters perform comparably to those with optimal hyperparameters, suggesting their insensitivity to hyperparameter settings.

REFERENCES

[1] Siddhant Arora. 2020. A survey on graph neural networks for knowledge graph completion. *arXiv preprint arXiv:2007.12374* (2020).

[2] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. *Advances in neural information processing systems* 29 (2016).

[3] Pablo Barceló, Floris Geerts, Juan Reutter, and Maksimilian Ryschkov. 2021. Graph neural networks with local graph parameters. *Advances in Neural Information Processing Systems* 34 (2021), 25280–25293.

[4] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. 2021. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910* (2021).

[5] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

[6] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 1 (2022), 657–668.

[7] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. 2018. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287* (2018).

[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[9] Jianfei Chen and Jun Zhu. 2018. Stochastic training of graph convolutional networks. (2018).

[10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. 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*. 257–266.

[11] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330, 4 (2003), 771–783.

[12] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2022. Graph Neural Networks with Learnable Structural and Positional Representations. In *International Conference on Learning Representations*.

[13] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems* 33 (2020), 22092–22103.

[14] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*.

[15] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. In *international conference on machine learning*. PMLR, 2083–2092.

[16] Wenhao Gao, Sai Pooja Mahajan, Jeremias Sulam, and Jeffrey J Gray. 2020. Deep learning in protein structural modeling and design. *Patterns* 1, 9 (2020).

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

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[19] William L Hamilton. 2020. *Graph representation learning*. Morgan & Claypool Publishers.

[20] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[21] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems* 31 (2018).

[22] Yinan Huang, Xingang Peng, Jianzhu Ma, and Muhan Zhang. 2022. Boosting the Cycle Counting Power of Graph Neural Networks with I^2-GNNs. *arXiv preprint arXiv:2210.13978* (2022).

[23] Shweta Ann Jacob, Paul Louis, and Amirali Salehi-Abari. 2023. Stochastic subgraph neighborhood pooling for subgraph classification. In *Proceedings of the 32nd ACM international conference on information and knowledge management*. 3963–3967.

[24] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[25] Boris Knyazev, Graham W Taylor, and Mohamed Amer. 2019. Understanding attention and generalization in graph neural networks. *Advances in neural information processing systems* 32 (2019).

[26] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*. PMLR, 3744–3753.

[27] AA Leman and Boris Weisfeiler. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya* 2, 9 (1968), 12–16.

[28] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems* 33 (2020), 4465–4478.

[29] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*. PMLR, 3835–3845.

[30] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[31] Paul Louis, Shweta Ann Jacob, and Amirali Salehi-Abari. 2022. Sampling enclosing subgraphs for link prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 4269–4273.

[32] Paul Louis, Shweta Ann Jacob, and Amirali Salehi-Abari. 2023. Simplifying subgraph representation learning for scalable link prediction. *arXiv preprint arXiv:2301.12562* (2023).

[33] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably powerful graph networks. *Advances in neural information processing systems* 32 (2019).

[34] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. 2018. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902* (2018).

[35] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*.

[36] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4602–4609.

[37] Alireza A Namanloo, Julie Thorpe, and Amirali Salehi-Abari. 2022. Improving Peer Assessment with Graph Neural Networks. *International Educational Data Mining Society* (2022).

[38] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. PMLR, 2014–2023.

[39] Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. 2021. DropGNN: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 21997–22009.

[40] Pál András Papp and Roger Wattenhofer. 2022. A theoretical comparison of graph neural network extensions. In *International Conference on Machine Learning*. PMLR, 17323–17345.

[41] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

[42] Danial Saber and Amirali Salehi-Abari. 2024. Scalable Expressiveness through Preprocessed Graph Perturbations. *arXiv preprint arXiv:2406.11714* (2024).

[43] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011).

[44] Zhihao Shi, Xize Liang, and Jie Wang. 2023. LMC: Fast training of GNNs via subgraph sampling with provable convergence. *arXiv preprint arXiv:2302.00924* (2023).

[45] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. 2003. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics* 19, 10 (2003), 1183–1193.

[46] Ameya Velingker, Ali Kemal Sinop, Ira Ktena, Petar Veličković, and Sreenivas Gollapudi. 2022. Affinity-aware graph networks. *arXiv preprint arXiv:2206.11941* (2022).

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

[48] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph neural networks in recommender systems: a survey. *Comput. Surveys* 55, 5 (2022), 1–37.

[49] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[50] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1365–1374.

[51] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 974–983.

[52] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable

pooling. *Advances in neural information processing systems* 31 (2018).
[53] Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 10737–10745.
[54] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. *Advances in neural information processing systems* 30 (2017).
[55] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).
[56] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
[57] Muhan Zhang and Pan Li. 2021. Nested graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 15734–15747.
[58] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2021. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems* 34 (2021), 9061–9073.
[59] Wentao Zhang, Zeang Sheng, Mingyu Yang, Yang Li, Yu Shen, Zhi Yang, and Bin Cui. 2022. NAFS: A Simple yet Tough-to-beat Baseline for Graph Representation Learning. In *International Conference on Machine Learning*. PMLR, 26467–26483.
[60] Yongqi Zhang and Quanming Yao. 2022. Knowledge graph reasoning with relational digraph. In *Proceedings of the ACM web conference 2022*. 912–924.
[61] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. 2022. From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness. In *International Conference on Learning Representations*.
[62] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems* 32 (2019).