Diffusion Probabilistic Models for Structured Node Classification

Hyosoon Jang [1] **Seunghyeon Park** [1] **Sangwoo Mo** [2] **Sungsoo Ahn** [1]

Abstract

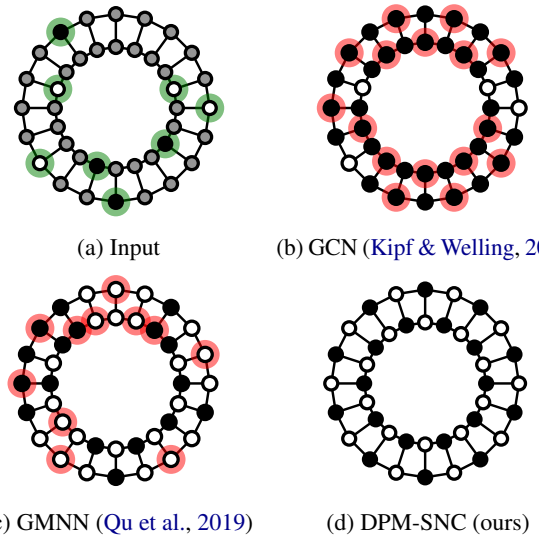
This paper studies structured node classification on graphs, where the predictions should consider dependencies between the node labels. In particular, we focus on solving the problem for partially labeled graphs where it is essential to incorporate the information in the known label for predicting the unknown labels. To address this issue, we propose a novel framework leveraging the diffusion probabilistic model for structured node classification (DPM-SNC). At the heart of our framework is the extraordinary capability of DPM-SNC to (a) learn a joint distribution over the labels with an expressive reverse diffusion process and (b) make predictions conditioned on the known labels utilizing manifold-constrained sampling. Since the DPMs lack training algorithms for partially labeled data, we design a novel training algorithm to apply DPMs, maximizing a new variational lower bound. We also theoretically analyze how DPMs benefit node classification by enhancing the expressive power of GNNs. We extensively verify the superiority of our DPM-SNC in diverse scenarios, which include not only the transductive setting but also the inductive setting.

1. Introduction

In this paper, we address the node classification problem, which is a fundamental problem in machine learning on graphs. One representative example is a transductive problem to classify documents from a partially labeled citation graph (Sen et al., 2008). Recently, graph neural networks (GNNs) (Kipf & Welling, 2016; Hamilton et al., 2017) have shown great success in this problem over their predecessors (Borgwardt & Kriegel, 2005; Grover & Leskovec, 2016). Their success stems from the high capacity of nonlinear neural architectures combined with message passing.

[1]POSTECH [2]KAIST. Correspondence to: Sungsoo Ahn <sungsoo.ahn@postech.ac.kr>.

Accepted to ICML workshop on Structured Probabilistic Inference & Generative Modeling, Honolulu, Hawaii, USA. PMLR 202, 2023.



(a) Input (b) GCN (Kipf & Welling, 2016) (c) GMNN (Qu et al., 2019) (d) DPM-SNC (ours)

Figure 1. Results of various methods to solve node classification on a non-attributed and partially labeled cyclic grid. (a) The task is to estimate $p(\boldsymbol{y}_U|\boldsymbol{y}_L, G)$ where the known labels $\boldsymbol{y}_L$ are highlighted in green and the unknown labels $\boldsymbol{y}_U$ are colored in gray. (b)-(d) Red highlights the incorrect predictions made by the corresponding algorithm. Conventional GNN (GCN) fails to make an informative prediction while our DPM-SNC perfectly predicts all the labels.

However, the conventional GNNs are incapable of *structured node classification*: predicting node labels while considering the node-wise label dependencies. That is, given a graph G with vertices $\mathcal{V}$ and node labels $\{y_i : i \in \mathcal{V}\}$, a GNN with parameter θ outputs an unstructured prediction, i.e., $p_\theta(y_i, y_j|G) = p_\theta(y_i|G)p_\theta(y_j|G)$ for $i, j \in \mathcal{V}$. Especially, this limitation becomes problematic in a transductive setting where the prediction can be improved by incorporating the known labels, e.g., output $p_\theta(y_i|G, y_j)$ for known y_j. In Figures 1(a) and 1(b), we elaborate on this issue with an example where the conventional GNN fails to make an informative prediction.

To resolve this issue, recent studies have investigated combining GNNs with classical structured prediction algorithms, i.e., schemes that consider the node label dependencies (Ma et al., 2019a;b; Graber & Schwing, 2019; Qu et al., 2022; Wang & Leskovec, 2020; Hang et al., 2021). They combine GNNs with conditional random fields (Lafferty et al., 2001), label propagation (Zhu & Ghahramani, 2002), or iterative classification algorithm (Sen et al., 2008). Despite

the promising outcomes demonstrated by these studies, their approach relies on the classical algorithms to express joint dependencies between node labels and may lack expressive power or consistency in incorporating known labels.

Contribution. We propose a novel framework for structured node classification called DPM-SNC. Our key idea is to leverage the diffusion probabilistic model (DPM) (Ho et al., 2020), motivated by two significant advantages for solving structured node classification: (a) it can effectively learn joint dependencies between node labels, and (b) it can easily incorporate conditions during inference via posterior sampling. Figure 1 highlights that DPM significantly outperforms previous methods when the prediction problem requires a complete understanding of label dependencies.

However, DPM cannot be directly applied to the transductive scenario, where the model needs to maximize its log-likelihood for partially labeled graphs. We propose a novel training algorithm to address this challenge. Our method maximizes a new variational lower bound of the marginal likelihood of the graph over the unlabeled nodes, involving the alternative optimization of DPM and a variational distribution. In addition, we provide a theoretical analysis that supports the benefits of DPM-SNC for node classification by enhancing the expressive power of GNNs.

We demonstrate the effectiveness of DPM-SNC on various datasets. In the transductive setting, we conduct experiments on synthetic and real-world benchmarks. In the inductive setting, we conduct experiments on inductive node classification and the algorithmic reasoning tasks (Du et al., 2022b). DPM-SNC outperformed baselines in both settings, effectively learning joint dependencies.

2. Diffusion Probabilistic Models for Structured Node Classification

In this section, we introduce the structured node classification problem (Section 2.1). We then explain how the diffusion probabilistic models (DPMs) offer a promising solution for this problem (Section 2.2).

2.1. Structured node classification

We address structured node classification on partially labeled graphs. Our problem involves a graph $G = (\mathcal{V}, \mathcal{E}, \boldsymbol{x})$ consisting of nodes $\mathcal{V}$, edges $\mathcal{E}$, and node attributes $\boldsymbol{x} = \{x_i : i \in \mathcal{V}\}$. We also denote the node labels by $\boldsymbol{y} = \{y_i : i \in \mathcal{V}\}$. We let $\mathcal{V}_L$ and $\mathcal{V}_U$ denote the set of labeled and unlabeled nodes, while $\boldsymbol{y}_L$ and $\boldsymbol{y}_U$ denote the corresponding labels for each set, e.g., $\boldsymbol{y}_L = \{y_i : i \in \mathcal{V}_L\}$. Our objective is to predict the unknown labels $\boldsymbol{y}_U$ by training on the partially labeled graph, aiming to infer the true conditional distribution $p(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$.

To address this problem while considering node label dependencies, structured prediction algorithms (Lafferty et al., 2001; Sen et al., 2008) try to solve two tasks: (a) learning a joint distribution $p_\theta(\boldsymbol{y}_U, \boldsymbol{y}_L|G)$ to maximize the likelihood of labeled data $p_\theta(\boldsymbol{y}_L|G)$ and (b) inferring from the distribution $p_\theta(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$ conditioned on known labels.

2.2. Diffusion probabilistic models for structured node classification

In this work, we consider diffusion probabilistic models for structured node classification (DPM-SNC) stems from their (a) high expressivity in learning a joint distribution over data and (b) the ability to easily infer from a posterior distribution conditioned on partially observed data, where both strengths benefit solving structured node classification.

To this end, we formally describe DPMs for a graph G associated with node-wise labels $\boldsymbol{y}$. At a high level, DPMs consist of two parts: forward and reverse processes. Given the number of diffusion steps T, the forward process constructs a sequence of noisy labels $\boldsymbol{y}^{(1:T)} = [\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(T)}]$ using a fixed distribution $q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y}^{(0)})$. Next, given an initial sample $\boldsymbol{y}^{(T)}$ sampled from $p(\boldsymbol{y}^{(T)})$, the reverse process $p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(0:T-1)}|\boldsymbol{y}^{(T)}, G)$ aims to recover the forward process. To be specific, the forward and the reverse process are factorized as $q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y}^{(0)}) = \prod_{t=1}^{T} q(\boldsymbol{y}^{(t)}|\boldsymbol{y}^{(t-1)})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(0:T-1)}|\boldsymbol{y}^{(T)}, G) = \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t-1)}|\boldsymbol{y}^{(t)}, G)$.

By leveraging shared latent variables $\boldsymbol{y}^{(t)}$ across multiple steps in the reverse process, the reverse process effectively considers the dependencies in the output. Also, DPMs can easily infer from a posterior distribution conditioned on partially observed data $p_{\boldsymbol{\theta}}(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$. Specifically, the incremental updating of $\boldsymbol{y}^{(t)}$ for $t = 1, \dots, T$ allows the DPM to incorporate the known label $\boldsymbol{y}_L$, e.g., applying projection with manifold-based corrections (Chung et al., 2022).

3. Training Diffusion Probabilistic Models on Partially Labeled Graphs

We introduce a novel training algorithm for DPM-SNC on partially labeled graphs, based on maximizing a variational lower bound for the log-likelihood of known labels.

Variational lower bound. At a high-level, our algorithm trains the DPM to maximize the log-likelihood of training data $\log p_\theta(\boldsymbol{y}_L|G)$, which is defined as follows:

$$\mathcal{L} = \log p_\theta(\boldsymbol{y}_L|G) = \log \sum_{\boldsymbol{y}_U} \sum_{\boldsymbol{y}^{(1:T)}} p_{\boldsymbol{\theta}}(\boldsymbol{y}_L, \boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|G).$$

However, this likelihood is intractable due to the exponentially large number of possible combinations for the unknown labels $\boldsymbol{y}_U$ and the noisy sequence of labels $\boldsymbol{y}^{(1:T)}$. To address this issue, we train the DPM based on a new

variational lower bound $\mathcal{L} \geq \mathcal{L}_{\text{VLB}}$, expressed as follows:

$$\mathcal{L}_{\text{VLB}} = \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)}\left[\mathbb{E}_{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G)}{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})}\right] - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L)\right].$$

Here, $q(\cdot)$ is a variational distribution which is factorized by $q(\boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|\boldsymbol{y}_L) = q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})q(\boldsymbol{y}_U|\boldsymbol{y}_L)$, where $\boldsymbol{y} = \boldsymbol{y}_U \cup \boldsymbol{y}_L$. We provide detailed derivation in Appendix A.1.

Training. To maximize the variational lower bound $\mathcal{L}_{\text{VLB}}$, we alternatively update the reverse process $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G)$ and the variational distribution $q(\boldsymbol{y}_U|\boldsymbol{y}_L)$, where the detailed parameterization is described in Appendix A.2. In particular, we train $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G)$ to maximize the Monte Carlo approximation of $\mathcal{L}_{\text{VLB}}$ by applying sampling to the variational distribution $q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})q(\boldsymbol{y}_U|\boldsymbol{y}_L)$.The detailed training objective is described in Appendix A.3.

Next, we set the variational distribution $q(\boldsymbol{y}_U|\boldsymbol{y}_L)$ to be the empirical distribution of $p_{\boldsymbol{\theta}}(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$ which can be inferred using the manifold-constrained sampling (Chung et al., 2022). This update is derived from the condition $q(\boldsymbol{y}_U|\boldsymbol{y}_L) = p_\theta(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$ being necessary to maximize $\mathcal{L}_{\text{VLB}}$, similar to the derivation of fixed-point iteration for optimization (Rhoades, 1991). We describe the detailed training procedure in Appendix A.4.

4. Theoretical Analysis

We demonstrate that DPMs have enhanced the expressive power for solving the graph isomorphism test compared to conventional GNNs, implying improved expressive power for node classification problems. We assess the expressive power of our DPM-SNC by introducing its analog, aggregated Weisfeiler-Lehman (AGG-WL) test. Then, with an analog of GNNs, i.e., 1-dimensional WL (1-WL) test (Weisfeiler & Leman, 1968), we introduce the following theorem.

Theorem 1. *Let 1-WL-GNN be a GNN as powerful as the 1-WL test. Then, DPM-SNC using a 1-WL-GNN is strictly more powerful than the 1-WL-GNN in distinguishing non-isomorphic graphs.*

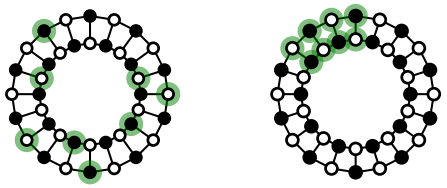
We provide the formal proof in Appendix B. We remark that our analysis can be easily extended to applying other latent variable models, e.g., variational auto-encoders (Kingma & Welling, 2013), for node classification.

5. Experiments

In the experiments, we consider the structured prediction baselines: : LP (Wang & Zhang, 2006), PTA (Dong et al., 2021), LPA (Wang & Leskovec, 2020), GMNN (Qu et al., 2019), G^{3}NN (Ma et al., 2019a), CLGNN (Hang et al., 2021), SPN (Qu et al., 2022).

We provide the details of our implementation for transductive and inductive setting in Appendices C.1 and C.2 respectively. For all experiments, we also describe the detailed data statics in Appendix D, and the detailed experimental setup in Appendix E.

5.1. Transductive setting



(a) Scattered nodes (b) Localized nodes

Figure 2. Illustration of two training nodes scenarios. Green highlights the training nodes, where the labels are known.

Table 2. The transductive node classification accuracy on synthetic data. **Bold** number indicates the best score.

Method	Scattered nodes	Localized nodes
GCN	50.13$_{\pm 0.66}$	51.32$_{\pm 1.42}$
+ GMNN	64.24$_{\pm 3.21}$	48.23$_{\pm 4.81}$
+ G^{3}NN	50.13$_{\pm 0.66}$	51.32$_{\pm 1.42}$
+ CLGNN	87.08$_{\pm 2.41}$	53.41$_{\pm 2.14}$
+ DPM-SNC	**98.56**$_{\pm 0.71}$	**90.91**$_{\pm 3.45}$

Synthetic data. First, we evaluate DPM-SNC on a $2 \times n$ non-attributed cyclic grid, where each node is assigned a binary label and neighboring nodes having different labels. Then, we consider two scenarios: one where the known labels are randomly scattered, and another where they are clustered in a local region. Both scenarios are illustrated in Figures 2(a) and 2(b). These two scenarios verify the capability for capturing both short and long-range dependencies between node labels.

Table 2 shows that our method significantly improves accuracy compared to the baselines. Furthermore, our method also excels in the localized labeled nodes scenario, while the other baselines fail. This highlights the superiority of DPM-SNC in considering label dependencies.

Homophilic graph. In real-world transductive node classification tasks, we first consider five homophilic graph datasets: Pubmed, Cora, and Citeseer (Yang et al., 2016); Photo and Computer (Shchur et al., 2018). For all the datasets, we evaluate the node-level accuracy, and the subgraph-level accuracy which measures the ratio of nodes with all neighboring nodes being correctly classified.

The results are presented in Table 1(a). Our method outperforms the structured prediction-specialized baselines in both node-label accuracy and subgraph-level accuracy. These results highlight the superiority of DPM-SNC in solving real-world node classification problems. Furthermore, even

Table 1. The transductive node classification performance. **Bold** numbers indicate (a) the best score among the structured prediction methods using the same GNN (b) the best score among all methods.

(a): Node-level accuracy (N-Acc.) and subgraph-level (Sub-Acc.) accuracy on homophilic graphs.

	Pubmed		Cora		Citeseer		Photo		Computer	
Method	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.
LP	69.1±0.0	45.7±0.0	68.1±0.0	46.9±0.0	46.1±0.0	29.8±0.0	81.0±2.0	37.2±1.7	69.9±2.9	15.1±1.1
PTA	80.1±0.2	55.2±0.4	82.9±0.4	62.6±0.8	71.3±0.4	51.4±0.7	91.1±1.5	51.0±1.5	81.6±1.7	26.3±1.0
GCN	79.7±0.3	55.8±0.6	81.4±0.8	59.3±1.1	70.9±0.8	49.8±0.6	91.0±1.2	52.0±1.0	82.4±1.5	27.0±1.5
+LPA	79.6±0.6	53.5±0.9	81.7±0.7	60.3±1.5	71.0±0.6	50.2±1.0	91.3±1.2	52.9±2.0	83.7±1.4	28.5±2.4
+GMNN	82.6±1.0	58.1±1.4	82.6±0.9	61.8±1.3	72.8±0.7	52.0±0.8	91.2±1.2	54.3±1.4	82.0±1.0	28.0±1.6
+G^3NN	80.9±0.7	56.9±1.1	82.5±0.4	62.3±0.8	73.9±0.7	53.1±1.0	90.7±1.1	53.0±2.0	82.1±1.2	28.1±2.1
+CLGNN	81.7±0.5	57.8±0.7	81.9±0.5	61.8±0.8	72.0±0.7	51.6±0.9	91.1±1.0	53.4±1.8	83.3±1.2	28.5±1.4
+DPM-SNC	**83.0**±0.9	**59.2**±1.2	**83.2**±0.5	**63.1**±0.9	**74.4**±0.5	**53.6**±0.6	**92.2**±0.8	**55.3**±2.1	**84.1**±1.3	**29.7**±1.8
GCNII	82.0±0.8	57.2±1.1	84.0±0.6	63.4±0.8	72.9±0.5	52.1±0.7	91.2±1.2	53.2±1.5	82.5±1.4	26.6±1.3
+DPM-SNC	**83.8**±0.7	**61.6**±0.9	**85.3**±0.6	**65.8**±0.7	**74.1**±0.5	**54.1**±0.9	**92.8**±1.1	**54.2**±1.2	**84.4**±1.8	**29.2**±1.1

(b): Node-level accuracy on heterophilic graphs.

	Empire	Rating
GCN	73.6±0.7	48.7±0.6
SAGE	85.7±0.6	53.6±0.3
GAT	80.8±0.3	49.0±0.6
GAT-sep	88.7±0.4	52.7±0.6
GT	86.5±0.7	51.1±0.6
GT-sep	87.3±0.3	52.1±0.8
DPM-SNC	**89.5**±0.4	**54.6**±0.3

Table 3. The inductive node classification performance. **Bold** numbers indicate the best score.

Method	Pubmed	Cora	Citeseer	PPI
GCN	54.5±0.5	59.6±0.5	49.8±0.4	99.1±0.0
+G^3NN	53.9±0.7	59.7±0.4	50.7±0.4	99.3±0.0
+CLGNN	53.9±0.5	60.2±0.3	50.5±0.3	99.2±0.0
+SPN	54.9±0.4	60.3±0.5	51.0±1.0	99.3±0.0
+DPM-SNC	**55.1**±0.4	**60.8**±0.3	**51.4**±0.5	**99.4**±0.0

when we combine our method with GCNII (Chen et al., 2020), our method achieves performance improvements. As can be observed, DPM-SNC consistently improves performance regardless of the backbone network. We provide the complete experiments table in Appendix F.1.

Heterophilic graph. To validate whether our framework can consider heterophily dependencies, we also consider recently proposed heterophilic graph datasets: Empire and Ratings (Platonov et al.), where most heterophily-specific GNNs fail to solve. In Table 1(b), we compare our method with base GNNs reported by Platonov et al..

We report our results in Table 1(b). Here, one can see that our method again achieves competitive performance on heterophilic graphs. These results stems from the capability for considering label dependencies, involving heterophily dependencies. We provide the complete experiments table in Appendix F.2.

5.2. Inductive setting

Inductive node classification. Following Qu et al. (2022), we construct small-scale graphs from Pubmed, Cora, and Citeseer, and construct large-scale graphs from PPI (Zitnik & Leskovec, 2017). We evaluate graph-level accuracy for small-scale graphs and micro-F1 score for large-scale graphs. The graph-level accuracy measures the ratio of graphs with where all the predictions are correct.

From Table 3, one can observe that our DPM-SNC shows competitive results. These results suggest that the DPM-SNC also solves inductive node classification well, thanks to their capability for learning the node label dependencies. We provide the complete experiments table in Appendix F.3.

Table 4. The graph algorithmic reasoning tasks performance on graph with ten nodes. **Bold** numbers indicate the best score.

Method	Copy	Connected	Shortest
Feedforward	0.3016	0.1796	0.1233
Recurrent	0.3015	0.1794	0.1259
Programmatic	0.3053	0.2338	0.1375
Iterative feedforward	0.6163	0.4908	0.4588
IREM	0.0019	0.1424	0.0274
DPM-SNC	**0.0011**	**0.0724**	**0.0138**

Algorithmic reasoning. We also evaluate DPM-SNC to predict the outcomes of graph algorithms. Here, we show that the capability of DPM-SNC to make a structured prediction even brings benefits to solving the graph algorithms reasoning tasks by incorporating a deep understanding between algorithmic elements. We evaluate the performance of our DPM-SNC on three graph algorithmic reasoning benchmarks proposed by Du et al. (2022b): edge copy, connected component, and shortest path. We report the performance using the mean square error. We also compare our method with five methods reported by Du et al. (2022b).

Table 4 shows that DPM-SNC achieves competitive results compared to the baselines. These results suggest that the DPM-SNC can easily solve algorithmic reasoning tasks thanks to its ability to make structured predictions. We provide the complete experiments table in Appendix F.4.

6. Conclusion

In this paper, we propose diffusion probabilistic models for solving structured node classification (DPM-SNC). Extensive experiments show that DPM-SNC outperforms existing structured node classification methods.

7. Acknowledgements

This work partly was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. IITP-2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)), the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2022R1C1C1013366), and Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2022R1A6A1A03052954).

References

Banino, A., Balaguer, J., and Blundell, C. Pondernet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning (AutoML)*. 22

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910*, 2021. 11

Bo, D., Wang, X., Shi, C., and Shen, H. Beyond low-frequency information in graph convolutional networks. In *AAAI*. AAAI Press, 2021. 21

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pp. 8–pp. IEEE, 2005. 1

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *International conference on machine learning*, pp. 1725–1735. PMLR, 2020. 4, 21

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=n6jl7fLxrP. 21

Chung, H., Sim, B., Ryu, D., and Ye, J. C. Improving diffusion models for inverse problems using manifold constraints. *arXiv preprint arXiv:2206.00941*, 2022. 2, 3, 8, 9

Dong, H., Chen, J., Feng, F., He, X., Bi, S., Ding, Z., and Cui, P. On the equivalence of decoupled graph convolution network and label propagation. In *Proceedings of the Web Conference 2021*, pp. 3651–3662, 2021. 3, 21

Du, L., Shi, X., Fu, Q., Ma, X., Liu, H., Han, S., and Zhang, D. Gbk-gnn: Gated bi-kernel graph neural networks for modeling both homophily and heterophily, 2022a. 21

Du, Y., Li, S., Tenenbaum, J., and Mordatch, I. Learning iterative reasoning through energy minimization. In *International Conference on Machine Learning*, pp. 5570–5582. PMLR, 2022b. 2, 4, 16, 18, 22

Graber, C. and Schwing, A. Graph structured prediction energy networks. *Advances in Neural Information Processing Systems*, 32, 2019. 1

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016. 1

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *NeurIPS*, 30, 2017. 1, 21

Hang, M., Neville, J., and Ribeiro, B. A collective learning framework to boost gnn expressiveness for node classification. In *International Conference on Machine Learning*, pp. 4040–4050. PMLR, 2021. 1, 3, 11, 21, 22

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020. 2, 7, 8

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019. 20

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 3

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 19, 21, 22

Lafferty, J., McCallum, A., and Pereira, F. C. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001. 1, 2

Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., and Qian, W. Finding global homophily in graph neural networks when meeting heterophily. *arXiv preprint arXiv:2205.07308*, 2022. 21

Ma, J., Tang, W., Zhu, J., and Mei, Q. A flexible generative framework for graph-based semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019a. 1, 3, 21, 22

Ma, T., Xiao, C., Shang, J., and Sun, J. CGNF: Conditional graph neural fields, 2019b. URL https://openreview.net/forum?id=ryxMX2R9YQ. 1

Maurya, S. K., Liu, X., and Murata, T. Improving graph neural networks with simple architecture design. *arXiv preprint arXiv:2105.07634*, 2021. 21

Murphy, R., Srinivasan, B., Rao, V., and Ribeiro, B. Relational pooling for graph representations. In *International Conference on Machine Learning*, pp. 4663–4673. PMLR, 2019. 12

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of gnns under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*. 4, 17, 19, 20, 21

Qu, M., Bengio, Y., and Tang, J. Gmnn: Graph markov neural networks. In *International conference on machine learning*, pp. 5241–5250. PMLR, 2019. 1, 3, 16, 21

Qu, M., Cai, H., and Tang, J. Neural structured prediction for inductive node classification. *arXiv preprint arXiv:2204.07524*, 2022. 1, 3, 4, 17, 22

Rhoades, B. Some fixed point iteration procedures. *International Journal of Mathematics and Mathematical Sciences*, 14(1):1–16, 1991. 3

Schwarzschild, A., Borgnia, E., Gupta, A., Huang, F., Vishkin, U., Goldblum, M., and Goldstein, T. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021. 22

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 1, 2

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. 3, 17

Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020. 21

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015. 22

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 16, 19

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ. 21, 22

Wang, F. and Zhang, C. Label propagation through linear neighborhoods. In *Proceedings of the 23rd international conference on Machine learning*, pp. 985–992, 2006. 3, 21

Wang, H. and Leskovec, J. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755*, 2020. 1, 3, 21

Wang, X. and Zhang, M. How powerful are spectral graph neural networks. In *International Conference on Machine Learning*, pp. 23341–23362. PMLR, 2022. 21

Weisfeiler, B. and Leman, A. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968. 3, 10

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 15

Yang, Z., Cohen, W., and Salakhudinov, R. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016. 3, 17

Zhang, M., Li, P., Xia, Y., Wang, K., and Jin, L. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021. 11

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020. 21

Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11168–11176, 2021. 21

Zhu, X. and Ghahramani, Z. Learning from labeled and unlabeled data with label propagation. 2002. 1

Zitnik, M. and Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33 (14):i190–i198, 2017. 4, 17

A. Details of DPM on partially labeled graphs

A.1. Derivation of variational lower bound

In this section, we provide a detailed derivation of the variational lower bound in **??**.

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_L|G) &= \log \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|G)} \left[p_{\boldsymbol{\theta}}(\boldsymbol{y}_L|G, \boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}) \right] \\
&= \log \mathbb{E}_{q(\boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|\boldsymbol{y}_L)} \left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G)}{q(\boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|\boldsymbol{y}_L)} \right] \\
&\geq \mathbb{E}_{q(\boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|\boldsymbol{y}_L)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G) - q(\boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|\boldsymbol{y}_L) \right] \\
&= \mathbb{E}_{q(\boldsymbol{y}_U, \boldsymbol{y}^{(1:T)}|\boldsymbol{y}_L)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G) - q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y}) - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right] \\
&= \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)} \left[\mathbb{E}_{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})} [\log p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G) - q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})] - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right].
\end{aligned}$$

A.2. Parameterization

We use a Gaussian diffusion (Ho et al., 2020) to parameterize the forward process $q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})$ as follows:

$$q(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(T)}|\boldsymbol{y}^{(0)}) = \prod_{t=1}^{T} \mathcal{N}(\boldsymbol{y}^{(t)}; \sqrt{1-\beta_t}\boldsymbol{y}^{(t-1)}, \beta_t \mathbf{I}),$$

where $\mathbf{I}$ is an identity matrix, $\beta_1, \ldots, \beta_T$ are fixed variance schedules. Here, we set the variance schedule to promote $q(\boldsymbol{y}^{(T)}|\boldsymbol{y}^{(0)}) \approx \mathcal{N}(\boldsymbol{y}^{(T)}; \mathbf{0}, \mathbf{I})$ by setting $\beta_t < \beta_{t+1}$ for $t = 0, \ldots, T-1$ and $\beta_T = 1$. We then parameterize initial distribution $p(\boldsymbol{y}^T)$ and reverse diffusion process $p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t-1)}|G, \boldsymbol{y}^{(t)})$ as $\mathcal{N}(\boldsymbol{y}^{(T)}; \mathbf{0}, \mathbf{I})$ and $\mathcal{N}(\boldsymbol{y}^{(t-1)}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t), \sigma_t^2)$, respectively. Here, we set σ_t^2 to β_t. We also define $\mu_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t)$ as follows:

$$\mu_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t) = \frac{1}{\sqrt{\alpha_t}} \left(\boldsymbol{y}^{(t)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t) \right), \tag{1}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. We implement the residual function $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t)$ through a GNN. Finally, we describe the distribution $q(\boldsymbol{y}_U|\boldsymbol{y}_L)$ as an empirical distribution over a fixed-size buffer $\mathcal{B}$ containing multiple estimates of the unknown labels $\boldsymbol{y}_U$. This buffer is updated throughout the training. The implementations are described in Appendix C.

A.3. Detailed training objective

We describe the detailed training objective of $\mathcal{L}_{\text{VLB}}$ for optimization. We first rewrite $\mathcal{L}_{\text{VLB}}$ as follows:

$$\begin{aligned}
\mathcal{L}_{\text{VLB}} &= \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)} \left[\mathbb{E}_{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G) - \log q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y}) \right] - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right] \\
&= \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)} \left[\mathbb{E}_{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})} \left[\sum_{t=1}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t-1)}|G, \boldsymbol{y}^{(t)})}{q(\boldsymbol{y}^{(t)}|\boldsymbol{y}^{(t-1)})} + \log p(\boldsymbol{y}^{(T)}) \right] - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right] \\
&= \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)} \left[\mathbb{E}_{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})} \left[\sum_{t=1}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t-1)}|G, \boldsymbol{y}^{(t)})}{q(\boldsymbol{y}^{(t-1)}|\boldsymbol{y}^{(t)}, \boldsymbol{y})} \frac{q(\boldsymbol{y}^{(t-1)}|\boldsymbol{y})}{q(\boldsymbol{y}^{(t)}|\boldsymbol{y})} + \log p(\boldsymbol{y}^{(T)}) \right] - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right] \\
&= \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)} \left[\mathbb{E}_{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})} \left[\sum_{t=1}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t-1)}|G, \boldsymbol{y}^{(t)})}{q(\boldsymbol{y}^{(t-1)}|\boldsymbol{y}^{(t)}, \boldsymbol{y})} + \log \frac{p(\boldsymbol{y}^{(T)})}{q(\boldsymbol{y}^{(T)}|\boldsymbol{y})} \right] - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right] \\
&= \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)} \left[\sum_{t=1}^{T} \mathbb{E}_{q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t-1)}|G, \boldsymbol{y}^{(t)})}{q(\boldsymbol{y}^{(t-1)}|\boldsymbol{y}^{(t)}, \boldsymbol{y})} \right] + \mathbb{E}_{q(\boldsymbol{y}^{(T)}|\boldsymbol{y})} \left[\log \frac{p(\boldsymbol{y}^{(T)})}{q(\boldsymbol{y}^{(T)}|\boldsymbol{y})} \right] - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right] \\
&= \mathbb{E}_{q(\boldsymbol{y}_U|\boldsymbol{y}_L)} \left[\sum_{t=1}^{T} \mathcal{L}_{\text{DPM}}^{(t)} + C - \log q(\boldsymbol{y}_U|\boldsymbol{y}_L) \right].
\end{aligned}$$

Here, $\mathcal{L}_{\text{DPM}}^{(t)}$ is a training objective for a t step, and C is a constant with respect to the parameters $\boldsymbol{\theta}$. Following Ho et al. (2020), we simplify $\mathcal{L}_{\text{DPM}}^{(t)}$ with a residual function $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t)$ in Equation (1).

$$\mathcal{L}_{\text{DPM}}^{(t)} = C - \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\boldsymbol{y} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, G, t)\right\|_2^2\right],$$

where C is a constant with respect to the parameters $\boldsymbol{\theta}$. An additional suggestion from Ho et al. (2020) is to set all weights of the mean squared error to one instead of $\beta_t^2/2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)$, and we follow this suggestion in this paper.

A.4. Training algorithm

Algorithm 1 Semi-supervised training algorithm of DPM-SNC

Input: Graph G, node attributes $\boldsymbol{x}$, known labels $\boldsymbol{y}_L$, buffer size K, the number of inserted samples N_1 at each iteration, and the number of training steps N_2 at each iteration.
Train a mean-field GNN $p_\phi(\boldsymbol{y}|G)$.
Initialize the buffer $\mathcal{B}$ by $p_\phi(\boldsymbol{y}_U|G)$.
repeat
  for $i = 1, \ldots, N_1$ **do**
    Get $\boldsymbol{y}_U \sim p_\theta(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$ using manifold-constrained sampling. ▷ *Appendix A.5*
    Update $\mathcal{B} \leftarrow \mathcal{B} \cup \{\boldsymbol{y}_U\}$.
    If $|\mathcal{B}| > K$, remove oldest one in $\mathcal{B}$.
  end for
  for $i = 1, \ldots, N_2$ **do**
    Sample $\boldsymbol{y}_U \sim \mathcal{B}$.
    Update θ to maximize $\mathcal{L}_{\text{VLB}}$ with G, and $\boldsymbol{y} = \boldsymbol{y}_L \cup \boldsymbol{y}_U$. ▷ *Appendix A.3*
  end for
until converged

In this section, we provide a detailed training algorithm that maximizes variational lower bound $\mathcal{L}_{\text{VLB}}$ with the parameterization specified in Appendix A.2. Specifically, we describe how our algorithm alternatively updates the reverse process $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G)$ and the variational distribution $q(\boldsymbol{y}_U|\boldsymbol{y}_L)$ defined by an empirical distribution over a buffer $\mathcal{B}$.

First, reverse process $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{y}^{(1:T)}|G)$ is trained to maximize Monte Carlo approximation of $\mathcal{L}_{\text{LVB}}$ by applying sampling to the variational distribution $q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})q(\boldsymbol{y}_U|\boldsymbol{y}_L)$, i.e., sampling $\boldsymbol{y}_U$ from the buffer $\mathcal{B}$ and applying the diffusion process $q(\boldsymbol{y}^{(1:T)}|\boldsymbol{y})$ to $\boldsymbol{y} = \boldsymbol{y}_U \cup \boldsymbol{y}_L$.

Next, we update the variational distribution $q(\boldsymbol{y}_U|\boldsymbol{y}_L)$ by inserting samples from the distribution $p_\theta(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$ into the buffer $\mathcal{B}$. Here, we use manifold-constrained sampling of DPM to infer from the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$ (Chung et al., 2022). The detailed sampling procedure is described in Appendix A.5. Furthermore, we initialize the buffer $\mathcal{B}$ with samples from a mean-field GNN $p_\phi(\boldsymbol{y}|G)$, which outputs an independent joint distribution over node labels, i.e., $p_{\boldsymbol{\theta}}(\boldsymbol{y}_U|G, \boldsymbol{y}_L) = \prod_{i\in\mathcal{V}_U} p_{\boldsymbol{\theta}}(\boldsymbol{y}_i|G)$. We describe the overall optimization procedure in Algorithm 1

A.5. Manifold-constrained sampling

Algorithm 2 Manifold-constrained sampling

Input: Graph G, node attributes $\boldsymbol{x}$, labels $\boldsymbol{y}_L$, and temperature of randomness τ^2.
Get $\boldsymbol{y}^{(T)} \sim \mathcal{N}(\boldsymbol{y}^{(T)}; \mathbf{0}, \tau^2\mathbf{I})$ ▷ *Initial sampling*
for $t = T-1, \dots, 0$ **do**
  Get $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \tau^2\mathbf{I})$
  Set $\tilde{\boldsymbol{y}}^{(t)} \leftarrow \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{y}^{(t+1)} - \frac{\beta_{t+1}}{\sqrt{1-\bar{\alpha}_{t+1}}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t+1)}, G, t+1)\right) + \sigma_{t+1}\boldsymbol{z}$
  Set $\hat{\boldsymbol{y}}^{(t+1)} \leftarrow \frac{1}{\sqrt{\bar{\alpha}_{t+1}}}\left(\boldsymbol{y}^{(t+1)} - \frac{1-\bar{\alpha}_{t+1}}{\sqrt{1-\bar{\alpha}_{t+1}}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t+1)}, G, t+1)\right)$
  Set $\bar{\boldsymbol{y}}^{(t)} \leftarrow \left(\tilde{\boldsymbol{y}}^{(t)} - \gamma\frac{\partial}{\partial \boldsymbol{y}^{(t+1)}}\left\|\boldsymbol{y}_L - \hat{\boldsymbol{y}}_L^{(t+1)}\right\|_2^2\right)$ ▷ *Manifold-constrained gradient*
  Set $\boldsymbol{y}_U^{(t)} \leftarrow \bar{\boldsymbol{y}}_U^{(t)}$
  Get $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \tau^2\mathbf{I})$
  Set $\boldsymbol{y}_L^{(t)} \leftarrow \sqrt{\bar{\alpha}_t}\boldsymbol{y}_L + \sqrt{1-\bar{\alpha}_t}\boldsymbol{z}$ ▷ *Projection step*
  Set $\boldsymbol{y}^{(t)} \leftarrow \boldsymbol{y}_L^{(t)} \cup \boldsymbol{y}_U^{(t)}$
end for
return $\boldsymbol{y}_U^{(0)}$

To sample $\boldsymbol{y}_U$ from $p_{\boldsymbol{\theta}}(\boldsymbol{y}_U|G, \boldsymbol{y}_L)$, we use a manifold-constrained sampling proposed by Chung et al. (2022). Here, the update rule of the reverse process for $t = 0, \dots, T-1$ is defined as follows:

$$\tilde{\boldsymbol{y}}^{(t)} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{y}^{(t+1)} - \frac{\beta_{t+1}}{\sqrt{1-\bar{\alpha}_{t+1}}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t+1)}, G, t+1)\right) + \sigma_{t+1}\boldsymbol{z}, \tag{2}$$

$$\boldsymbol{y}_U^{(t)} = \bar{\boldsymbol{y}}_U^{(t)}, \qquad \bar{\boldsymbol{y}}^{(t)} = \tilde{\boldsymbol{y}}^{(t)} - \gamma\frac{\partial}{\partial \boldsymbol{y}^{(t+1)}}\left\|\boldsymbol{y}_L - \hat{\boldsymbol{y}}_L^{(t+1)}\right\|_2^2, \tag{3}$$

$$\boldsymbol{y}_L^{(t)} = \sqrt{\bar{\alpha}_t}\boldsymbol{y}_L + \sqrt{1-\bar{\alpha}_t}\boldsymbol{z}, \tag{4}$$

where $\boldsymbol{z}$ is sampled from $\mathcal{N}(\boldsymbol{z}; \mathbf{0}, \tau^2\mathbf{I})$. The Equation (2) is a temporal reverse diffusion step before applying the manifold-constrained samplings. The Equation (3) applies the manifold-constrained gradient $\gamma\frac{\partial}{\partial \boldsymbol{y}^{(t+1)}}\|\boldsymbol{y}_L - \hat{\boldsymbol{y}}_L^{(t+1)}\|_2^2$. Here, $\hat{\boldsymbol{y}}^{(t+1)}$ is the label estimate in $t+1$ steps defined as follows:

$$\hat{\boldsymbol{y}}^{(t+1)} = \frac{1}{\sqrt{\bar{\alpha}_{t+1}}}\left(\boldsymbol{y}^{(t+1)} - \frac{1-\bar{\alpha}_{t+1}}{\sqrt{1-\bar{\alpha}_{t+1}}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t+1)}, G, t+1)\right),$$

where γ is a hyper-parameter. We set γ to $1/\|\boldsymbol{y}_L - \hat{\boldsymbol{y}}_L^{(t+1)}\|_2^2$. The Equation (4) is a projection step. Additionally, we introduce a parameter τ to control the randomness; when we set τ to zero, the modified reverse step becomes deterministic. This allows us to control the randomness in obtaining samples. We describe the detailed sampling algorithm in Algorithm 2.

B. WL test and GNN's expressiveness

In this section, we provide the proof of Theorem 1 in detail.

B.1. Preliminaries

Algorithm 3 1-dimensional Weisfeiler-Lehman algorithm

Input: Graph $G = (\mathcal{V}^G, \mathcal{E}^G, \boldsymbol{x}^G)$ and the number of iterations T.
Output: Color mapping $\chi_G : \mathcal{V}^G \to \mathcal{C}$.
Initialize: Let $\chi_G^0(v) \leftarrow \mathrm{hash}(x_v)$ for $v \in \mathcal{V}^G$.
for $t = 1, \dots, T$ **do**
  for each $v \in \mathcal{V}^G$ **do**
    Set $\chi_G^t(v) \leftarrow \mathrm{hash}(\chi_G^{t-1}(v), \{\!\{\chi_G^{t-1}(u) : u \in \mathcal{N}_G(v)\}\!\})$
  end for
end for
Return: χ_G^T

We denote set as $\{\}$, and multiset as $\{\!\{\}\!\}$, which is a set allowing duplicate elements. We represent the cardinality of set or multiset $\mathcal{S}$ as $|\mathcal{S}|$. A graph is denoted as $G = (\mathcal{V}^G, \mathcal{E}^G, \boldsymbol{x}^G)$, where $\mathcal{V}^G$ stands for the set of nodes, $\mathcal{E}^G$ for the set of edges, and $\boldsymbol{x}^G = \{x_v^G : v \in \mathcal{V}^G\}$ for the node-wise attributes. We also use $\mathcal{N}_G(v) \coloneqq \{u \in \mathcal{V}^G : \{v, u\} \in \mathcal{E}^G\}$ to denote the set of neighbor nodes of node v in graph G. We abbreviate a set of integers using the notation $[m] \coloneqq \{0, \dots, m\}$. We also assume the hash functions are all injective and denote them by $\mathrm{hash}(\cdot)$.

Definition 1 (Vertex coloring)**.** Vertex coloring $\chi_G(v)$ is an injective hash function that maps a vertex v in graph G to a color c from an abstract color set $\mathcal{C}$. Next, with a slight abuse of notation, we let the graph color χ_G indicate the multiset of node colors in graph G, i.e., $\chi_G \coloneqq \{\!\{\chi_G(v) : v \in \mathcal{V}^G\}\!\}$.

Now we explain the 1-dimensional Weisfeiler-Lehman (1-WL) algorithm (Weisfeiler & Leman, 1968), a classical algorithm to distinguish non-isomorphic graphs. At a high level, the 1-WL test iteratively updates node colors based on their neighbors until a stable coloring is reached. To be specific, given a vertex v, the initial node color $\chi_G^0(v)$ is set using an injective hash function on the node attribute x_v. At each iteration, each node color is refined based on the aggregation of neighbor node colors, e.g., $\{\!\{\chi_G^{t-1}(u) : u \in \mathcal{N}_G(v)\}\!\}$. At the final T-th iteration, the algorithm returns the graph color χ_G^T. We provide a detailed description in Algorithm 3.

The 1-WL test allows to compare a pair of graphs G, H using the refined graph colors χ_G^T, χ_H^T. If the two graph colors χ_G^T and χ_H^T are not equivalent, two graphs G, H are guaranteed to be non-isomorphic. Otherwise, the test is inconclusive, i.e., the two graphs G, H are possibly isomorphic. We provide an example run of the 1-WL test in Figure 3(a), where the test is inconclusive.

Related to the 1-WL algorithm, we first prove a characteristic of it that will be later used in our proof.

Lemma 1. *Consider running 1-WL on two graphs $G = (\mathcal{V}^G, \mathcal{E}^G, \boldsymbol{x}^G)$ and $H = (\mathcal{V}^H, \mathcal{E}^H, \boldsymbol{x}^H)$. If the initial graph colors of two graphs are distinct, i.e., $\chi_G^0 \neq \chi_H^0$, the respective outputs of the 1-WL are also distinct, i.e., $\chi_G^T \neq \chi_H^T$.*

Proof. We first prove that $\chi_G^t \neq \chi_H^t$ is satisfied when $\chi_G^{t-1} \neq \chi_H^{t-1}$. The rest of the proof is straightforward by induction on t. To be specific, the 1-WL updates χ_G^t and χ_H^t as follows:

$$\chi_G^t = \{\!\{\mathrm{hash}(\chi_G^{t-1}(v), \{\!\{\chi_G^{t-1}(u) : u \in \mathcal{N}_G(v)\}\!\}) : v \in \mathcal{V}^G\}\!\}$$
$$\chi_H^t = \{\!\{\mathrm{hash}(\chi_H^{t-1}(v), \{\!\{\chi_H^{t-1}(u) : u \in \mathcal{N}_H(v)\}\!\}) : v \in \mathcal{V}^H\}\!\}.$$

Since $\mathrm{hash}(\cdot)$ is an injective function, χ_G^t and χ_H^t are distinct for $\chi_G^{t-1} \neq \chi_H^{t-1}$. □

B.2. AGG-WL test

Next, we describe the newly proposed AGG-WL test, which is an analog of our DPM-SNC. Similar to the 1-WL test, our AGG-WL test assigns node colors and iteratively refines them based on the neighbor node colors. However, AGG-WL

generates augmented graphs at initialization, creating multiple "views" on the graph with diverse initial vertex coloring. Then it applies 1-WL on each of the augmented graphs to obtain the augmented graph colors. Finally, AGG-WL aggregates the augmented graphs colors.

The complete algorithm is described in Algorithm 4. Given a graph $G = (\mathcal{V}^G, \mathcal{E}^G, \boldsymbol{x}^G)$ and the set of possible node-wise augmentations $\mathcal{Z}^{\mathcal{G}}$, the algorithm generates augmented graph G^m. In particular, the augmented graph G^m is defined as follows:

$$G^m = (\mathcal{V}^G, \mathcal{E}^G, \tilde{\boldsymbol{x}}^{G^m}), \quad \tilde{\boldsymbol{x}}^{G^m} = \{(x_v^G \| \, z_v^m) : v \in \mathcal{V}\}, \quad \boldsymbol{z}^m = \{z_v^m : v \in \mathcal{V}\}, \quad \boldsymbol{z}^m \in \mathcal{Z}^G$$

where $\boldsymbol{z}^m$ is an augmentation method in $\mathcal{Z}^G$. The algorithm creates an augmented graph G^m by node-wise concatenation of $\boldsymbol{z}^{\boldsymbol{m}}$ to its node attributes $\boldsymbol{x}^G$, where $\tilde{\boldsymbol{x}}^{G^m}$ denotes the node-wise attribute augmented by $\boldsymbol{z}^m$. The symbol $\cdot\|\cdot$ denotes the concatenation of two elements. Furthermore, $\boldsymbol{z}^0$ is a unique token, where G^0 has the same information as G. Also, with a slight abuse of notation, we let the graph color returned by the AGG-WL χ_G^{AGG} indicate the multiset of augmented graph colors, i.e., $\chi_G^{\text{AGG}} := \{\!\{\chi_{G^m}^T : m \in [|\mathcal{Z}^G|]\}\!\}$. Here, $|\mathcal{Z}^G|$ denotes the cardinality of set $\mathcal{Z}^G$.

Algorithm 4 Aggregation Weisfeiler-Lehman algorithm

Input: Graph $G = (\mathcal{V}^G, \mathcal{E}^G, \boldsymbol{x}^G)$, the number of iterations T, and the augmentation set $\mathcal{Z}^G$.
Output: Color mapping $\chi_G : \mathcal{V}^G \to \mathcal{C}$.
Initialize: Generate $|\mathcal{Z}^G|$ augmented graphs $G^m = (\mathcal{V}^G, \mathcal{E}^G, \boldsymbol{x}^{G^m})$ where $\tilde{\boldsymbol{x}}^{G^m} = \{(x_v^G \| \, z_v^m) : v \in \mathcal{V}^G\}$ for $\boldsymbol{z}^m \in \mathcal{Z}^G$. Let $\chi_{G^m}^0(v) \leftarrow \text{hash}(\tilde{\boldsymbol{x}}_v^{G^m})$ for $v \in \mathcal{V}^G, m \in [|\mathcal{Z}^G|]$.
for $t = 1, \ldots, T$ **do**
  for each $v \in V$ **do**
    for $m \in [|\mathcal{Z}^G|]$ **do**
      $\chi_{G^m}^t(v) \leftarrow \text{hash}(\chi_{G^m}^{t-1}(v), \{\!\{\chi_{G^m}^{t-1}(u) : u \in \mathcal{N}_{G^m}(v)\}\!\})$
    end for
  end for
end for
$\chi_G^{\text{AGG}} \leftarrow \{\!\{\chi_{G^m}^T : m \in [|\mathcal{Z}^G|]\}\!\}$
Return: χ_G^{AGG}

Our contribution is establishing the connection between the AGG-WL and the DPM-SNC: aggregation of refined graph colors for augmented graphs and marginalization over latent variables. More detail between DPM-SNC and the AGG-WL test is described in Appendix B.3.2.

We note that our algorithm bears some similarities with several previous studies. The DS-WL and DSS-WL test (Bevilacqua et al., 2021) defined a modified WL-test based on the set of subgraphs (instead of augmented graphs) with a modified edge set $\mathcal{E}'$. Hang et al. (2021) uses a collective algorithm to consider pseudo-labels as additional inputs to boost GNN expressiveness. However, they rely on the assumption that one can find an "optimal" pseudo-label to discriminate a pair of graphs, which may be hard to realize in practice. Zhang et al. (2021) proposed labeling tricks that learns to capture dependence between nodes also by adding additional features, but mainly focus on link prediction in inductive settings.

B.3. Proof of Theorem 1

Let us start by restating Theorem 1.

Theorem 1. *Let 1-WL-GNN be a GNN as powerful as the 1-WL test. Then, DPM-SNC using a 1-WL-GNN is strictly more powerful than the 1-WL-GNN in distinguishing non-isomorphic graphs.*

Proof. We divide the proof into two parts, Appendices B.3.1 and B.3.2. In Appendix B.3.1, we show that the AGG-WL test is strictly more powerful than the 1-WL test. In Appendix B.3.2, we show that DPM-SNC using a 1-WL-GNN is as powerful as the AGG-WL test. From these two proofs, one can conclude that DPM-SNC using a 1-WL-GNN is strictly more powerful than 1-WL-GNNs in distinguishing non-isomorphic graphs. □

B.3.1. EXPRESSIVENESS OF AGG-WL TEST

In this subsection, we prove that the AGG-WL test is strictly more powerful that the 1-WL test. The high-level idea is showing that (i) the AGG-WL test is at least as powerful as the 1-WL test, i.e., any two graphs distinguishable by the 1-WL test is also distinguishable by the AGG-WL test, and (ii) there exist two non-isomorphic graphs that are indistinguishable by the 1-WL test, but distinguishable by our AGG-WL test.

Lemma 2. *AGG-WL is at least as powerful as WL in distinguishing non-isomorphic graphs, i.e., any two non-isomorphic graphs G, H distinguishable by WL are also distinguishable by AGG-WL.*

Proof. We first note that both 1-WL and AGG-WL can discriminate two non-isomorphic graphs with distinct sizes in a straightforward way. Therefore, we focus on non-trivial cases for two graphs $G = (\mathcal{V}^G, \mathcal{E}^G, \boldsymbol{x}^G), H = (\mathcal{V}^H, \mathcal{E}^H, \boldsymbol{x}^H)$ where the number of nodes and edges are same, i.e., $|\mathcal{V}^G| = |\mathcal{V}^H|, |\mathcal{E}^G| = |\mathcal{E}^H|$. The cardinality of the augmentation sets are also the same, $|\mathcal{Z}^G| = |\mathcal{Z}^H|$. We prove if these two graphs G, H are distinguishable by the 1-WL, two graphs G, H are also distinguishable by the AGG-WL, i.e., if $\chi^T_G \neq \chi^T_H$, then $\chi^{\text{AGG}}_G \neq \chi^{\text{AGG}}_H$.

First, since $\chi^T_G \neq \chi^T_H$ is satisfied, $\chi^T_{G^0} \neq \chi^T_{H^0}$ is trivial. Additionally, the initial graphs colors of graph augmented by the unique token and any other augmentations are distinct, i.e., $\chi^0_{G^0} \neq \chi^0_{H^1}, \ldots, \chi^0_{H^{|\mathcal{Z}^H|}}$. This implies $\chi^T_{G^0} \notin \chi^{\text{AGG}}_H = \{\chi^T_{H^0}, \ldots, \chi^T_{H^{|\mathcal{Z}^H|}}\}$ according to Lemma 1 which shows distinct initial graph colors produce the distinct outputs of 1-WL. Since $\chi^T_{G^0} \in \chi^{\text{AGG}}_G$, one can see that $\chi^{\text{AGG}}_G \neq \chi^{\text{AGG}}_H$. □

Next, we show that there exist two graphs G, H indistinguishable by the 1-WL test, but distinguishable by the AGG-WL test. We first show a simple example from Figure 3(a), then for a family of circular skip link (CSL) graphs (Murphy et al., 2019).

Lemma 3. *Two graphs A, B are indistinguishable by the 1-WL, but distinguishable by the AGG-WL.*

Proof. In Figure 3(a), two graphs A, B with 6 nodes are given. Here, the graph color computed by the 1-WL are equivalent, i.e., $\chi^T_G = \chi^T_H$ (two light blue nodes and four gray nodes). Therefore, two graphs A, B are indistinguishable by the 1-WL.

Next, we prove that two graphs A, B are distinguishable by the AGG-WL. Since Lemma 1 shows 1-WL always computes a distinct graph color from distinct initial graph colors, we only need to show that there exists augmented graphs A^m, B^m with the same initial color has distinct graph colors computed by the 1-WL. Then, for two augmented graph sets $\{A^m : m \in [|\mathcal{Z}^{\mathcal{V}^A}|]\}$ and $\{B^m : m \in [|\mathcal{Z}^{\mathcal{V}^B}|]\}$, we can show that the graph colors computed by AGG-WL are always distinct.

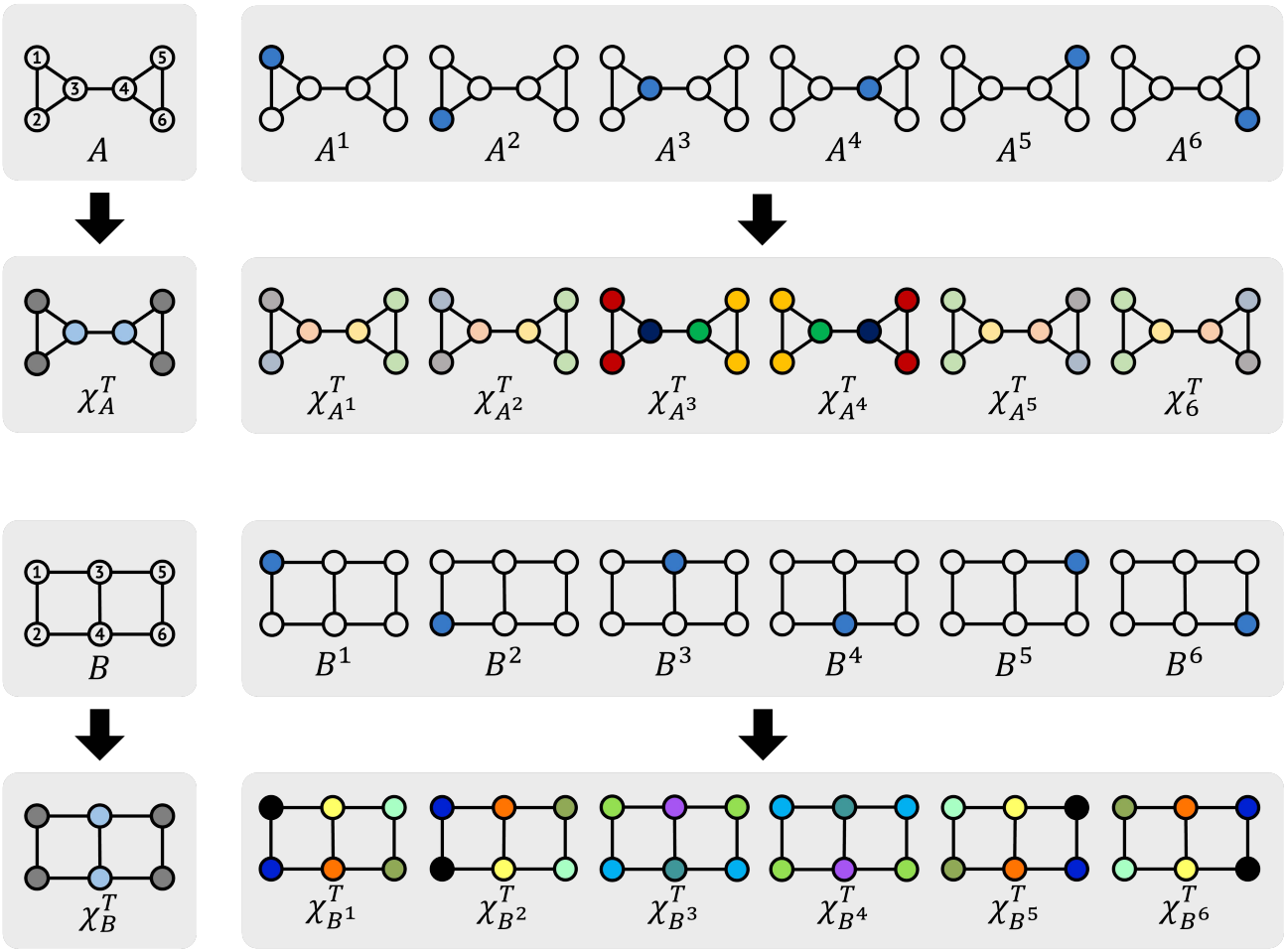
We show a simple case, where augmented graphs with non-zero augmentation on one node results distinct graph colors computed by the AGG-WL. We denote "single binary node feature augmentation", $\boldsymbol{z}^i$ as the case where node i is non-zero augmented and other nodes are augmented with zero, e.g., $\boldsymbol{z}^3 = \{0, 0, 1, 0, 0, 0\}$. In Figure 3(b) the non-zero augmented node i is colored in blue, and others are colored transparent. After applying the 1-WL algorithm on each graph, we obtain the graph color $\chi^T_{A^i}$ and $\chi^T_{B^i}$ for $i \in \{1, \ldots, 6\}$. It is clear that $\{\!\{\chi^T_{A^i} : i \in \{1, \ldots, 6\}\}\!\} \neq \{\!\{\chi^T_{B^i} : i \in \{1, \ldots, 6\}\}\!\}$, thereby two graphs A, B are distinguishable by AGG-WL. □

Circular skip link graphs. CSL graph is denoted as $\text{CSL}(n, r)$, where n is the number of nodes, .i.e., $V = \{0, \ldots, n-1\}$ and r is the skip connection length. For n and r, $r < n - 1$ must hold. There exists $2n$ edges, between node i and $(i+1) \mod n$ forming a cycle, and between nodes i and $(i+r) \mod n$ forming a skip link for $i \in \{0, \ldots, n-1\}$.

Lemma 4. *For $n \geq 8, r \in [3, n/2 - 1]$, two graphs CSL$(n, 2)$ and CSL(n, r) are indistinguishable by the 1-WL, but distinguishable by the AGG-WL.*

Proof. We first give a brief proof of why $\text{CSL}(n, 2)$ and $\text{CSL}(n, r)$ are indistinguishable by the 1-WL, then prove they are distinguishable by AGG-WL. We consider a non-trivial case where the node attributes are all same, i.e., $\chi^0_G(v) = \chi^0_G(u)$ for $v, u \in \mathcal{V}^G$.

Let the initial color be c_0 for all the nodes for graphs $\text{CSL}(n, 2)$ and $\text{CSL}(n, r)$, $\chi^0_G(v) = c_0 \; \forall v \in \mathcal{V}^G, G \in \{\text{CSL}(n, 2), \text{CSL}(n, r)\}$. Then, for all nodes $v \in \mathcal{V}^G$, the color refinement process can than be written as $\chi^1_G(v) = \text{hash}(c_0, \{\!\{c_0, c_0, c_0, c_0\}\!\})$. All nodes have identical colors c_1 for both graphs $\text{CSL}(n, 2), \text{CSL}(n, r)$, refining the initial graph color. Therefore $\text{CSL}(n, 2)$ and $\text{CSL}(n, r)$ are indistinguishable by the 1-WL.



(a) 1-WL (b) AGG-WL

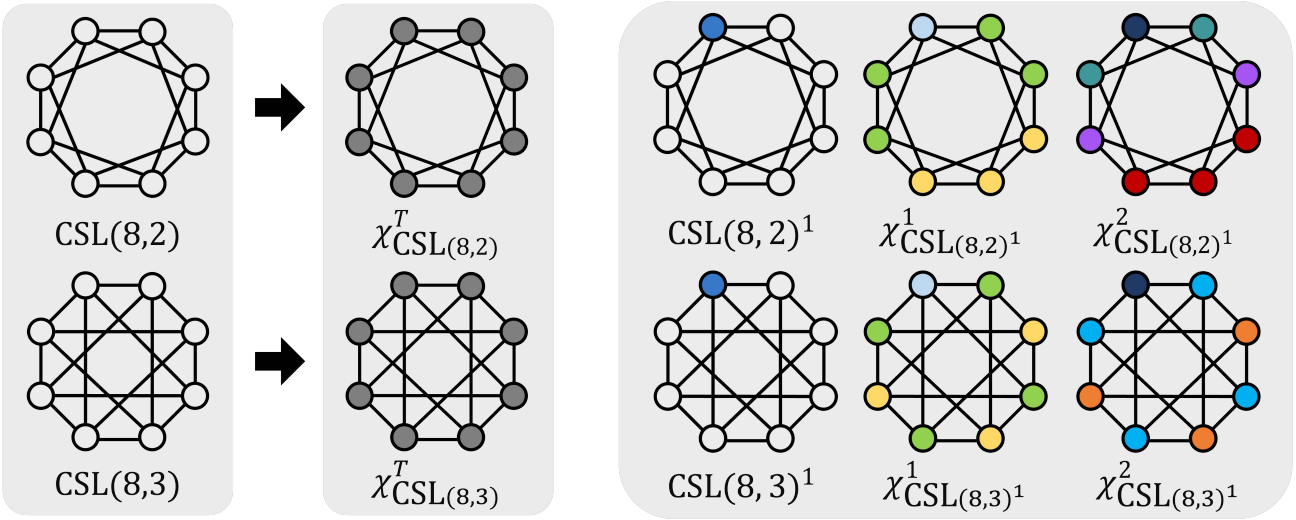
Figure 3. An example of two non-isomorphic graphs A, B indistinguishable by 1-WL, but distinguishable by the AGG-WL. (a) Two graphs A, B are associated with nodes sharing the same attribute. The 1-WL computes the same graph color for A, B, i.e., $\chi_A^T = \chi_B^T$. (b) Applying AGG-WL with "single binary node feature augmentation" to the graphs A and B. This results in six cases for each graph. One can see that the multiset of augmented graph colors are different, i.e., $\{\!\{\chi_{A^i}^T : i \in \{1, \dots, 6\}\}\!\} \neq \{\!\{\chi_{B^i}^T : i \in \{1, \dots, 6\}\}\!\}$, thereby two graphs A, B are distinguishable by the AGG-WL.

Now we consider the AGG-WL. Again, we only need to show that there exists augmented graphs A^i, B^j with the same initial color but having distinct graph colors refined by the 1-WL. Given symmetry, there is only one case of an augmented graph that adds a non-zero augmentation to one node. We use v_i to denote the i-th node in graph and denote the augmented node as v_0. We denote each augmented graph as $\text{CSL}(n, 2)^1, \text{CSL}(n, r)^1$, and let the initial color $\chi_G^0(v_0) = c_1$, and $\forall i \in \{1, \dots, n-1\}, \forall G \in \{\text{CSL}(n, 2)^1, \text{CSL}(n, r)^1\}$ $\chi_G^0(v_i) = c_0$.

Iteration 1. We focus on the four nodes connected to v_0. Since $r \in [3, n/2 - 1]$, two node v_{n-r}, v_r are distinct. The color refinement can be written as following:

- For $v \in \{v_1, v_2, v_{n-1}, v_{n-2}\}$, $\chi^1_{\text{CSL}(n,2)^1}(v) = \text{hash}(c_0, \{\!\{c_1, c_0, c_0, c_0\}\!\}) = c_2$.
- For $v \notin \{v_1, v_2, v_{n-1}, v_{n-2}\}$, $\chi^1_{\text{CSL}(n,2)^1}(v) = \text{hash}(c_0, \{\!\{c_0, c_0, c_0, c_0\}\!\}) = c_3$.
- For $v \in \{v_1, v_r, v_{n-1}, v_{n-r}\}$, $\chi^1_{\text{CSL}(n,r)^1}(v) = \text{hash}(c_0, \{\!\{c_1, c_0, c_0, c_0\}\!\}) = c_2$.
- For $v \notin \{v_1, v_r, v_{n-1}, v_{n-r}\}$, $\chi^1_{\text{CSL}(n,r)^1}(v) = \text{hash}(c_0, \{\!\{c_0, c_0, c_0, c_0\}\!\}) = c_3$.

Since 1-WL showed $\chi^1_{\text{CSL}(n,2)} = \chi^1_{\text{CSL}(n,r)}$ and augmented graphs showed $\chi^1_{\text{CSL}(n,2)^1} = \chi^1_{\text{CSL}(n,r)^1}$, two graphs are indistinguishable.



(a) Original graph and 1-WL coloring

(b) AGG-WL coloring on $\text{CSL}(8,2)^1, \text{CSL}(8,3)^1$

Figure 4. Example on two non-attribute $\text{CSL}(8,2), \text{CSL}(8,3)$ graph. (a) 1-WL returns same graph color for two graphs, i.e., $\chi^T_{\text{CSL}(8,2)} = \chi^T_{\text{CSL}(8,3)}$. (b) Considering symmetry, there is only one type of augmented graph that with non-zero augmentation on one node, we denoted the augmented graph as $\text{CSL}(8,2)^1$ and $\text{CSL}(8,3)^1$. In the process of 1-WL assigning colors to augmented graph, they result different graph color from the second iteration, and one can conclude $\chi^T_{\text{CSL}(8,2)^1} \neq \chi^T_{\text{CSL}(8,3)^1}$. Therefore $\text{CSL}(8,2), \text{CSL}(8,3)$ are distinguishable by the AGG-WL.

Iteration 2. Again, we focus on four nodes connected to v_0. Here, we describe color refinement for nodes $v_1, v_{n-1}, v_2, v_{n-2}$, since the following is enough to prove two graph colors are distinct.

- For $v \in \{v_1, v_{n-1}\}$, $\chi^2_{\text{CSL}(n,2)^1}(v) = \text{hash}(c_2, \{\!\{c_1, c_2, c_2, c_3\}\!\}) = c_4$.
- For $v \in \{v_2, v_{n-2}\}$, $\chi^2_{\text{CSL}(n,2)^1}(u) = \text{hash}(c_2, \{\!\{c_1, c_2, c_3, c_3\}\!\}) = c_5$.
- For $v \in \{v_1, v_{n-1}\}$, $\chi^2_{\text{CSL}(n,r)^1}(v) = \text{hash}(c_2, \{\!\{c_1, c_3, c_3, c_3\}\!\}) = c_6$.

Since $c_6 \notin \chi^2_{\text{CSL}(n,2)^1}$, two graphs are distinguishable, i.e., $\chi^t_{\text{CSL}(n,2)^1} \neq \chi^t_{\text{CSL}(n,r)^1}$.

Then $\chi^T_{\text{CSL}(n,2)^1} \neq \chi^T_{\text{CSL}(n,r)^1}$ is satisfied as proved in Lemma 1, thereby two graphs $\text{CSL}(n,2)$ and $\text{CSL}(n,r)$ are distinguishable by the AGG-WL. □

In Figure 4, we provide an example with $\text{CSL}(8,2)$ and $\text{CSL}(8,3)$, indistinguishable by the 1-WL but distinguishable the AGG-WL.

B.3.2. Correspondence between DPM-SNC and AGG-WL

Now, we explain the connection between the DPM-SNC and the AGG-WL test. At a high-level, we show how DPM-SNC simulates the color refinement process of the AGG-WL. Our main idea stems from the marginalization in DPM-SNC. As a latent variable model, it can define the probability of the graph color χ with the marginalization over augmented graphs, i.e., $p_\theta(\chi|G) = \int p_\theta(\chi|G, \boldsymbol{z}) p(\boldsymbol{z}|G) d\boldsymbol{z}$.[1] Specifically, DPM-SNC considers sample space of graph colors, where each member χ is obtained from $p_\theta(\chi|G, \boldsymbol{z})$ with an augmented graph G^m represented by $(G, \boldsymbol{z}^m)$. By construction, DPM-SNC considers graph colors refinements for multiple augmented graphs, similar to how AGG-WL works.

Lemma 5. *DPM-SNC using a 1-WL-GNN is as powerful as the AGG-WL test in distinguishing non-isomorphic graphs.*

[1]The node labels $\boldsymbol{y}$ is replaced to the graph color χ, and applying a latent variable is interpreted as applying an augmentation to the given graph.

Proof. We show that if two graphs G, H are distinguishable by AGG-WL test, it is also distinguishable by the DPM-SNC. Assume that the two graphs G, H are distinguishable by the AGG-WL. Then, the following condition is satisfied.

$$\chi_G^{\text{AGG}} = \{\!\{\chi_{G^m}^T : m \in [|\mathcal{Z}^G|]\}\!\} \neq \chi_H^{\text{AGG}} = \{\!\{\chi_{H^m}^T : m \in [|\mathcal{Z}^H|]\}\!\},$$

where the AGG-WL produces distinct sets of refined graph colors for G, H. Next, we discuss how the DPM-SNC can distinguish G, H. To this end, we first assume a GNN which is as powerful as the 1-WL test (Xu et al., 2018), denoted as 1-WL-GNN. Specifically, we denote χ_G^{GNN} as a graph color refined by the 1-WL-GNN, where $\chi_{G^m}^T \neq \chi_{H^n}^T$ implies $\chi_{G^m}^{\text{GNN}} \neq \chi_{H^n}^{\text{GNN}}$ for any augmented graph pair G^m, H^n. Then, under the $\chi_G^{\text{AGG}} \neq \chi_H^{\text{AGG}}$, the following condition is also satisfied.

$$\{\!\{\chi_{G^m}^{\text{GNN}} : m \in [|\mathcal{Z}^G|]\}\!\} \neq \{\!\{\chi_{H^m}^{\text{GNN}} : m \in [|\mathcal{Z}^H|]\}\!\}.$$

Here, we assume that DPM-SNC utilizes this 1-WL-GNN to output a graph color conditioned on an augmented graph, i.e., $p_\theta(\chi|G, \boldsymbol{z})$. Then, we can connect $p_\theta(\chi|G, \boldsymbol{z}^m)$ with $\chi_{G^m}^{\text{GNN}}$ as follows:

$$p_\theta(\chi|G, \boldsymbol{z}^m) = \mathbb{I}[\chi = \chi_{G^m}^{\text{GNN}}].$$

where $\mathbb{I}[a = b]$ is an indicator function whose value is 1 if $a = b$ and 0 otherwise. Next, we also assume that $p(\boldsymbol{z}|G)$ is a uniform distribution over $\mathcal{Z}^G$. Then one can show that:

$$p_\theta(\chi|G) = \frac{1}{|\mathcal{Z}^G|} \sum_{m \in [|\mathcal{Z}^G|]} \mathbb{I}[\chi = \chi_{G^m}^{\text{GNN}}].$$

Then it follows that $p_\theta(\chi|G) \neq p_\theta(\chi|H)$ since $\{\!\{\chi_{G^m}^{\text{GNN}} : m \in [|\mathcal{Z}^G|]\}\!\} \neq \{\!\{\chi_{H^m}^{\text{GNN}} : m \in [|\mathcal{Z}^H|]\}\!\}$. Therefore, G, H are distinguishable by DPM-SNC. □

Our proof is valid when considering the multiple outputs. However, our practical implementation of DPM-SNC does not use multiple outputs at inference time since the DPM only considers the multiple random variables in the training objective.[2] To this end, in Appendix F.5, we also investigate the inference scheme which aggregates multiple outputs to make final predictions.[3]

[2] Our inference method is described in Appendix C.1

[3] In practice, we also use a Gaussian diffusion for defining $p(\boldsymbol{z}|G)$, which still allows the GNN to consider infinite augmentations.

C. Implementation

In this section, we provide more details on how we implement the DPM-SNC for experiments.

C.1. Transductive settings

Here, we provide the detailed implementation of DPM-SNC for transductive node classification.

Model architecture. We parameterize the residual function $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t)$ of reverse diffusion step using a L-layer message-passing GNN as follows:

$$\begin{aligned}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}, G, t) &= g(\boldsymbol{h}^{(L)}),\\ h_i^{(\ell)} &= (\mathrm{COMBINE}^{(\ell)}(h_i^{(\ell-1)}, a_i^{(\ell)}) + f(t)) \| y_i^{(t)}),\\ a_i^{(\ell)} &= \mathrm{AGGREGATE}^{(\ell)}(\{h_j^{(\ell-1)} | (i,j) \in \mathcal{E}\}),\end{aligned}$$

where $g(h^{(L)})$ is a multi-layer perceptron that estimates the residual using the final node representation. AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$ functions are identical to the backbone GNN, and $\cdot \| \cdot$ indicates the concatenation. Here, h_i^0 is $x_i \| y_i^{(t)}$. The $f(\cdot)$ is a sinusoidal positional embedding function (Vaswani et al., 2017). We fix the dimension of sinusoidal positional embedding to 128.

Buffer construction. Following the temperature annealing approach of Qu et al. (2019) in sampling pseudo-labels for optimization, we also control the temperature of randomness in obtaining $\boldsymbol{y}_U$ from $p_{\boldsymbol{\theta}}(\boldsymbol{y}_U | G, \boldsymbol{y}_{\boldsymbol{L}})$ for buffer construction. To be specific, we use the variance multiplied by the temperature $\tau \in [0, 1]$, instead of the original variance in the reverse diffusion step, e.g., setting τ to zero makes the deterministic sampling.

Inference. To make final predictions $\boldsymbol{y}_U$ from $p_{\boldsymbol{\theta}}(\boldsymbol{y}_U | G, \boldsymbol{y}_L)$ for evaluation, we eliminate the randomness in inference time, i.e., set temperature τ to zero.[4] If the target is one-hot relaxation of discrete labels, we also discretize the final prediction by choosing a dimension with maximum value.

C.2. Inductive setting

Here, we provide the details of DPM-SNC for inductive node classification and graph algorithmic reasoning. For the inductive node classification, we use the same model architecture as in the transductive setting and use the deterministic inference strategy. For the graph algorithmic, we modify DPM-SNC to perform edge-wise prediction.

Graph algorithmic reasoning. Since the targets of the graph algorithmic reasoning task are defined on the edge-level, we apply a diffusion process to the edge labels. We then recover the edge-level noisy labels through the reverse process.

The denoising model architecture in the reverse process has a similar architecture to the model architecture of IREM (Du et al., 2022b). Specifically, the noisy edge target $\boldsymbol{y}^{(t)}$ is updated as follows. First, the noisy edge labels $\boldsymbol{y}^{(t)}$ and edge features are concatenated and passed through to the GNN layer, which aggregates them to obtain the node representation. Next, we apply element-wise addition of the time embedding vector to the node representation. Then, we concatenate a pair of node representations and noisy targets for the given edges and then apply a two-layer MLP to update edge labels.

In contrast to the node classification, we maintain randomness at inference time, i.e., we use the stochastic reverse process for obtaining edge labels. This approach is consistent with the IREM, which also includes randomness at inference time.

[4] We also investigate various stochastic inference strategies in Appendix F.5

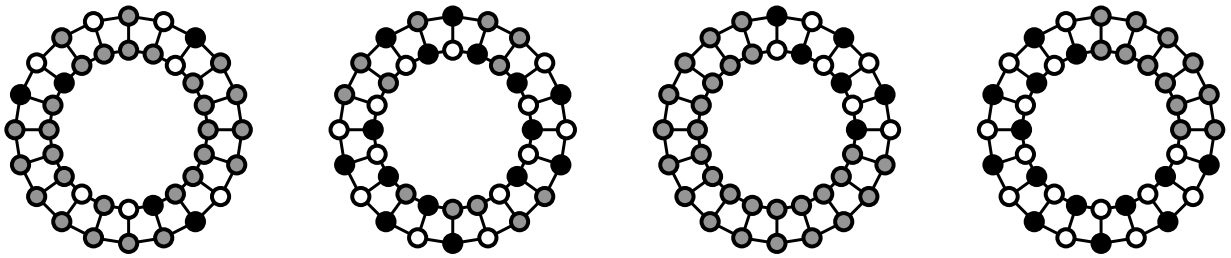
D. Data statistics

D.1. Synthetic data

We generate 1000×2 non-attributed cyclic grid and 100×2 non-attributed cyclic grid for scattered and localized training nodes scenarios, respectively. Then, we split 30%, 30%, and 40% of the entire nodes into training, validation, and test nodes.

- *Scattered training nodes*: We randomly sample nodes in the graph to split them into training, validation, and test nodes.[5]
- *Localized training nodes*: We select the nodes in the region within the 30×2 grid as training nodes. Then, we randomly sample the remaining nodes in the graph to split them into validation and test nodes.

For illustrative purposes, we also describe both scenarios in Figure 5 with smaller graphs.



(a) Scattered-training (b) Scattered-test/validation (c) Localized-training (d) Localized-test/validation

Figure 5. Illustration of two scenarios. The non-gray nodes represent nodes in each split.

D.2. Transductive node classification datasets

Table 5. The data statistics of transductive node classification datasets.

Dataset	♯ nodes	♯ edges	♯ features	♯ classes	♯ (training/validation/test) nodes
Pubmed (Yang et al., 2016)	19717	44338	500	3	(60/500/1000)
Cora (Yang et al., 2016)	2708	5429	1433	7	(140/500/1000)
Citeseer (Yang et al., 2016)	3327	4732	3703	6	(120/500/1000)
Photo (Shchur et al., 2018)	7487	119043	745	8	(160/240/7087)
Computers (Shchur et al., 2018)	13381	34493	767	10	(200/300/12881)
Roman (Platonov et al.)	22662	32927	300	18	(11331/5665/5666)
Ratings (Platonov et al.)	24492	93050	300	5	(12246/6123/6123)

Here, we consider a graph with partially labeled nodes. We describe the data statistics in Table 5.

D.3. Inductive node classification datasets

Table 6. The data statistics of inductive node classification datasets.

			(training/validation/test) data		
Dataset	♯ features	♯ classes	♯ graphs	Avg. ♯ nodes	Avg. ♯ edges
Pubmed (Qu et al., 2022)	500	3	(60/500/1000)	(6.0/5.4/5.6)	(6.7/5.8/6.7)
Cora (Qu et al., 2022)	1433	7	(140/500/1000)	(5.6/4.9/4.7)	(7.0/5.8/5.3)
Citeseer (Qu et al., 2022)	3703	6	(120/500/1000)	(4.0/3.8/3.8)	(4.3/4.0, 3/8)
PPI (Zitnik & Leskovec, 2017)	500	3	(20/2/2)	(2245.3/3257.0/2762.0)	(61318.4/99460.0/80988.0)

Here, we consider datasets consists of a set of graphs. We describe the data statistics in Table 6.

[5]Additionally, we also consider training with an additional 2×20 cyclic grid for visualization in Figure 1.

D.4. Graph algorithmic reasoning datasets

Following Du et al. (2022b), we generate training graphs in each training step. Here, the training graphs are composed of graphs of varying sizes, ranging from two to ten nodes. The node features are initialized to zero, and the labels are defined on the edges, e.g., the shortest distance between two nodes. Then, we evaluate performance on graphs with ten nodes. Furthermore, we also use graphs with 15 nodes to evaluate generalization capabilities.

E. Experiments setup

In this section, we describe the detailed experimental setup. For all experiments, we use a single GPU of NVIDIA GeForce RTX 3090. The hyper-parameters for each experiment are described in the following subsections.

E.1. Synthetic dataset

In this experiment, we implement each method with a one-layer GCN with 16 hidden dimensions. We search the learning rate within $\{1e{-}3, 5e{-}3, 1e{-}2\}$ for all methods. Other hyper-parameters of each method follow their default settings. For DPM-SNC, we fix the diffusion step to 100. We also set the size of the buffer to 50 and insert five samples into the buffer for every 30 training step. We use a pre-trained mean-field GNN until the buffer is updated 10 times. We report the performance with ten different random seeds.

E.2. Transductive node classification

Table 7. The hyper-parameter search ranges for the homophilic graph. For hyper-parameters without a specific method in parentheses, it applies to all methods in the respective category.

Method	Hyper-parameters	Search range
All methods	learning rate	$\{1e{-}3, 5e{-}3, 1e{-}2\}$
	weight decay	$\{1e{-}3, 5e{-}3, 1e{-}2\}$
GNN-based methods (LPA, GMNN, G^{3}NN, CLGNN, DPM-SNC)	number of layers	$\{2, 4\}$
	hidden dimension	$\{64, 128\}$
	weight of constraints for structured-prediction (LPA, G^{3}NN)	$\{0.1, 1.0, 10.0\}$
	pseudo-labels sampling temperature (GMNN, CLGNN, DPM-SNC)	$\{0.1, 0.3, 1.0\}$
Non-GNN methods (LP, PTA)	number of label propagation	$\{10, 100\}$
	hidden dimension (PTA)	$\{64, 128\}$
	damping factor	$\{0.1, 0.3, 0.5\}$

Homophilc graph. We describe the hyper-parameter search ranges in Table 7. For the GNN-based methods, we parameterize each method with GCN (Kipf & Welling, 2016) or GAT (Vaswani et al., 2017). Additionally, we apply dropout with $p = 0.5$ except for LP. Other hyper-parameters of each method follow their default settings. For DPM-SNC, we fix the diffusion step to 80. We also set the size of the buffer to 50 and insert five samples into the buffer for every 100 training step. We use a pre-trained mean-field GNN until the buffer is updated 20 times. We report the performance with ten different seeds.

Heterophilic graph. We describe the hyper-parameters for DPM-SNC as we use the numbers reported by Platonov et al. for baselines. We parameterize DPM-SNC with GAT-sep (Platonov et al.). We describe the hyper-parameter search ranges in Table 8. Additionally, we apply dropout with $p = 0.5$, and we fix the diffusion step to 80. We also set the size of the buffer to 50 and insert five samples into the buffer for every 100 training step. We use a pre-trained mean-field GNN until the buffer is updated 100 times. We report the performance with ten different seeds.

Table 8. The hyper-parameter search ranges of DPM-SNC for the heterophilic graph.

Hyper-parameters	Search range
learning rate	$\{3e{-}5, 1e{-}4, 3e{-}4\}$
weight decay	$\{0, 1e{-}5, 1e{-}4\}$
number of layers	$\{2, 4\}$
hidden dimension	$\{256, 512\}$
sampling temperature	$\{0.1, 0.3, 1.0\}$

E.3. Inductive node classification

Table 9. The hyper-parameter search ranges of all methods for the inductive node classificaiton.

Hyper-parameters	Search range
learning rate	$\{1e{-}3, 5e{-}3, 1e{-}2\}$ for small-scale graphs and $\{3e{-}5, 1e{-}4, 3e{-}4\}$ for huge-scale graphs
weight decay	$\{1e{-}3, 5e{-}3, 1e{-}2\}$ for small-scale graphs and $\{0, 1e{-}5, 1e{-}4\}$ for huge-scale graphs
number of layers	$\{2, 4\}$
hidden dimension	$\{64, 128\}$ for small-scale graphs and $\{512, 1024\}$ for huge-scale graphs

We describe the hyper-parameter search ranges in Table 9. In this experiements, we parameterize each method with GCN (Kipf & Welling, 2016) or GAT (Vaswani et al., 2017). For the small-scale graph datasets, i.e., Pubmed, Cora, and Citeseer, we apply dropout with $p = 0.5$. For the huge-scale graph datasets, i.e., PPI, we include the linear skip connection between

each GNN layer. Other hyper-parameters of each method follow their default settings. For DPM-SNC, we fix the diffusion step to 80. We report the performance with ten and five different seeds for small-scale and large-scale graphs, respectively.

E.4. Algorithmic reasoning

Here, we describe the hyper-parameter settings for DPM-SNC as we use the numbers reported by Platonov et al. for baselines. We search the learning rate and weight decay within $\{1\mathrm{e}{-4}, 3\mathrm{e}{-4}, 1\mathrm{e}{-3}\}$ and $\{0, 1\mathrm{e}{-5}, 1\mathrm{e}{-4}\}$, respectively. The hyper-parameters of the model are the same as the model implementation of IREM, using a three-layer GINEConv (Hu et al., 2019) with a 128 hidden dimension. We fix the diffusion step to 80.

F. Additional experiments

F.1. Transductive node classification on homophilic graphs.

Table 10. The transductive node classification performance. N-Acc. and Sub-Acc. denote the node-level and subgraph-level accuracy, respectively. **Bold** numbers indicate the best score among the structured prediction methods using the same GNN.

	Pubmed		Cora		Citeseer		Photo		Computer	
Method	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.	N-Acc.	Sub-Acc.
LP (Wang & Zhang, 2006)	69.1±0.0	45.7±0.0	68.1±0.0	46.9±0.0	46.1±0.0	29.8±0.0	81.0±2.0	37.2±1.7	69.9±2.9	15.1±1.1
PTA (Dong et al., 2021)	80.1±0.2	55.2±0.4	82.9±0.4	62.6±0.8	71.3±0.4	51.4±0.7	91.1±1.5	51.0±1.5	81.6±1.7	26.3±1.0
GCN (Kipf & Welling, 2016)	79.7±0.3	55.8±0.6	81.4±0.8	59.3±1.1	70.9±0.8	49.8±0.6	91.0±1.2	52.0±1.0	82.4±1.5	27.0±1.5
+LPA (Wang & Leskovec, 2020)	79.6±0.6	53.5±0.9	81.7±0.7	60.3±1.5	71.0±0.6	50.2±1.0	91.3±1.2	52.9±2.0	83.7±1.4	28.5±2.4
+GMNN (Qu et al., 2019)	82.6±1.0	58.1±1.4	82.6±0.9	61.8±1.3	72.8±0.7	52.0±0.8	91.2±1.2	54.3±1.4	82.0±1.0	28.0±1.6
+G^{3}NN (Ma et al., 2019a)	80.9±0.7	56.9±1.1	82.5±0.4	62.3±0.8	73.9±0.7	53.1±1.0	90.7±1.1	53.0±2.0	82.1±1.2	28.1±2.1
+CLGNN (Hang et al., 2021)	81.7±0.5	57.8±0.7	81.9±0.5	61.8±0.8	72.0±0.7	51.6±0.9	91.1±1.0	53.4±1.8	83.3±1.2	28.5±1.4
+DPM-SNC (ours)	**83.0**±0.9	**59.2**±1.2	**83.2**±0.5	**63.1**±0.9	**74.4**±0.5	**53.6**±0.6	**92.2**±0.8	**55.3**±2.1	**84.1**±1.3	**29.7**±1.8
GAT (Veličković et al., 2018)	79.1±0.5	55.8±0.5	81.5±0.6	61.3±0.9	71.0±0.8	50.8±1.0	90.8±1.0	50.8±1.9	83.1±1.6	27.8±2.2
+LPA (Wang & Leskovec, 2020)	78.7±1.1	56.0±1.2	81.5±0.9	60.7±0.8	71.3±0.9	50.1±0.9	91.3±0.8	52.7±2.1	**84.4**±1.0	29.4±2.6
+GMNN (Qu et al., 2019)	79.6±0.8	57.0±0.7	82.3±0.7	62.2±0.8	71.7±0.9	51.4±0.9	91.4±1.0	53.1±1.6	83.3±2.0	29.1±1.8
+G^{3}NN (Ma et al., 2019a)	77.9±0.4	55.9±0.5	82.7±1.3	62.7±1.3	74.0±0.8	53.7±0.5	91.5±0.9	52.6±2.2	83.1±1.7	28.8±2.4
+CLGNN (Hang et al., 2021)	80.0±0.6	57.5±1.2	81.8±0.6	61.5±0.9	72.1±0.8	52.1±0.8	90.6±0.8	51.9±1.8	82.6±1.2	28.4±1.8
+DPM-SNC (ours)	**81.7**±0.8	**59.0**±1.1	**83.8**±0.7	**63.8**±0.7	**74.3**±0.7	**54.0**±0.9	**92.0**±0.8	**54.0**±2.4	84.2±1.2	**30.0**±2.0
GCNII (Chen et al., 2020)	82.0±0.8	57.2±1.1	84.0±0.6	63.4±0.8	72.9±0.5	52.1±0.7	91.2±1.2	53.2±1.5	82.5±1.4	26.6±1.3
+DPM-SNC (ours)	**83.8**±0.7	**61.6**±0.9	**85.3**±0.6	**65.8**±0.7	**74.1**±0.5	**54.1**±0.9	**92.8**±1.1	**54.2**±1.2	**84.4**±1.8	**29.2**±1.1

We report the full experiments table in Table 10.

F.2. Transductive node classification on heterophilic graphs.

Table 11. The transductive node classification accuracy on heterophilic graphs. **Bold** numbers indicate the best score.

	Empire	Rating
H_2GCN (Zhu et al., 2020)	60.11±0.52	36.47±0.23
CPGNN (Zhu et al., 2021)	63.96±0.62	39.79±0.77
GPR-GNN (Chien et al., 2021)	64.85±0.27	44.88±0.34
FSGNN (Maurya et al., 2021)	79.92±0.56	52.74±0.83
GloGNN (Li et al., 2022)	59.63±0.69	36.89±0.14
FAGCN (Bo et al., 2021)	65.22±0.56	44.12±0.30
GBK-GNN (Du et al., 2022a)	74.57±0.47	45.98±0.71
JacobiConv (Wang & Zhang, 2022)	71.14±0.42	43.55±0.48
GCN (Kipf & Welling, 2016)	73.69±0.74	48.70±0.63
SAGE (Hamilton et al., 2017)	85.74±0.67	53.63±0.39
GAT (Veličković et al., 2018)	80.87±0.30	49.09±0.63
GAT-sep (Platonov et al.)	88.75±0.41	52.70±0.62
GT (Shi et al., 2020)	86.51±0.73	51.17±0.66
GT-sep (Platonov et al.)	87.32±0.39	52.18±0.80
DPM-SNC (ours)	**89.52**±0.46	**54.66**±0.39

We report the full experiments table in Table 11.

F.3. Inductive node classification.

Table 12. The inductive node classification performance. N-Acc., G-Acc., and F1 denote the node-level accuracy, graph-level accuracy, and micro-F1 score, respectively. **Bold** numbers indicate the best score among the structured prediction methods using the same GNN.

	Pubmed		Cora		Citeseer		PPI
Method	N-Acc.	G-Acc.	N-Acc.	G-Acc.	N-Acc.	G-Acc.	F1
GCN (Kipf & Welling, 2016)	80.25±0.42	54.58±0.51	83.36±0.43	59.67±0.51	76.37±0.35	49.84±0.47	99.15±0.03
+G^{3}NN (Ma et al., 2019a)	80.32±0.30	53.93±0.71	83.60±0.25	59.78±0.47	76.34±0.37	50.76±0.47	99.33±0.02
+CLGNN (Hang et al., 2021)	80.22±0.45	53.98±0.54	83.45±0.34	60.24±0.38	75.71±0.40	50.51±0.38	99.22±0.04
+SPN (Qu et al., 2022)	**80.78**±0.34	54.91±0.40	83.85±0.60	60.35±0.57	76.25±0.48	51.02±1.06	99.35±0.02
+DPM-SNC (ours)	80.58±0.41	**55.16**±0.43	**84.09**±0.27	**60.88**±0.36	**77.01**±0.49	**51.44**±0.56	**99.46**±0.02
GAT (Veličković et al., 2018)	80.10±0.45	54.38±0.54	79.71±1.41	56.66±1.40	74.91±0.22	49.87±0.44	99.54±0.01
+G^{3}NN (Ma et al., 2019a)	79.88±0.62	54.66±0.29	81.19±0.45	58.68±0.38	75.45±0.26	50.86±0.46	99.56±0.01
+CLGNN (Hang et al., 2021)	80.23±0.40	54.51±0.36	81.38±0.55	58.81±0.61	75.45±0.36	50.66±0.45	99.55±0.01
+SPN (Qu et al., 2022)	79.95±0.34	**54.82**±0.33	81.61±0.31	59.17±0.31	75.41±0.35	51.04±0.53	99.46±0.02
+DPM-SNC (ours)	**80.26**±0.37	54.26±0.47	**81.79**±0.46	**59.55**±0.49	**76.46**±0.60	**52.05**±0.71	**99.63**±0.01

We report the full experiments table in Table 12.

F.4. Graph algorithm reasoning.

Table 13. Performance on graph algorithmic reasoning tasks. **Bold** numbers indicate the best score. Same-MSE and Large-MSE indicate the performance on ten, and 15 nodes, respectively.

	Edge copy		Connected components		Shortest path	
Method	Same-MSE	Large-MSE	Same-MSE	Large-MSE	Same-MSE	Large-MSE
Feedforward	0.3016	0.3124	0.1796	0.3460	0.1233	1.4089
Recurrent (Schwarzschild et al., 2021)	0.3015	0.3113	0.1794	0.2766	0.1259	0.1083
Programmatic (Banino et al.)	0.3053	0.4409	0.2338	3.1381	0.1375	0.1290
Iterative feedforward (Sohl-Dickstein et al., 2015)	0.6163	0.6498	0.4908	1.2064	0.4588	0.7688
IREM (Du et al., 2022b)	0.0019	**0.0019**	0.1424	0.2171	0.0274	0.0464
DPM-SNC (ours)	**0.0011**	0.0038	**0.0724**	**0.1884**	**0.0138**	**0.0286**

We report the full experiments table in Table 13.

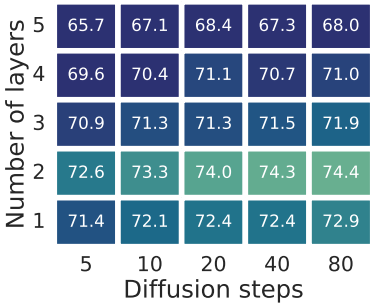


Figure 6. Accuracy with varying GNN layers and diffusion steps.

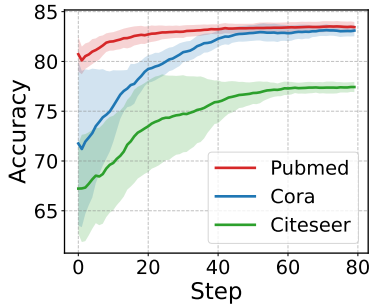


Figure 7. Accuracy for changes in diffusion steps.

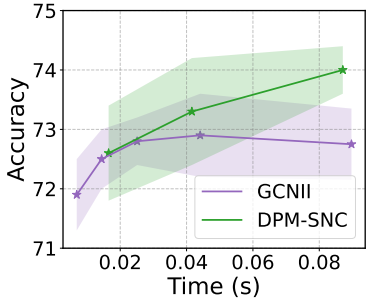
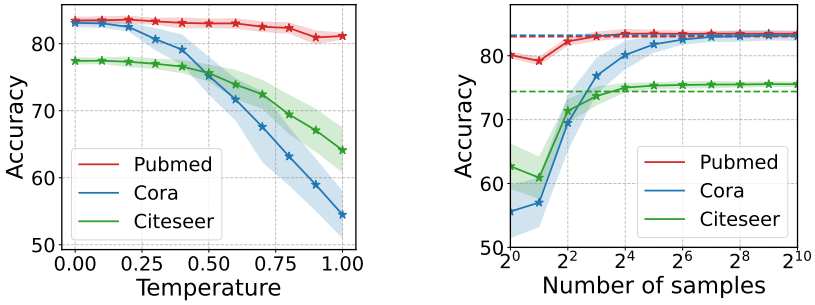


Figure 8. Accuracy with varying inference time.



(a) (b)

Figure 9. (a) Accuracy with varying temperature. (b) Accuracy with the varying number of samples. The dashed line represents the accuracy of the deterministic inference scheme.

F.5. Ablation stuides

Diffusion steps vs. number of GNN layers. We first verify that the performance gains in DPM-SNC mainly stem from the reverse diffusion process which learns the joint dependency between labels. To this end, we vary the number of diffusion steps along with the number of GNN layers. We report the corresponding results in Figure 6. One can observe that increasing the number of diffusion steps provides a non-trivial improvement in performance, which cannot be achieved by just increasing the number of GNN layers.

Accuracy over diffusion steps. We investigate whether the iteration in the reverse process progressively improves the quality of predictions. In Figure 7, we plot the changes in node-level accuracy in the reverse process as the number of iterations increases. The results confirm that the iterative inference process gradually increases accuracy, eventually reaching convergence.

Running time vs. performance. Here, we investigate whether the DPM-SNC can make a good trade-off between running time and performance. In Figure 8, we compare the change in accuracy of DPM-SNC with GCNII over the inference time on Citeseer by changing the number of layers and diffusion steps for DPM-SNC and GCNII, respectively. The backbone network of DPM-SNC is a two-layer GCN. One can observe that our DPM-SNC shows competitive performances compared to the GCNII at a similar time. Also, while increasing the inference times of the GCNII does not enhance performance, DPM-SNC shows further performance improvement.

Ablations on various inference schemes. We also study various inference strategies for our DPM-SNC. We first investigate how temperature control affects label inference in real-world node classification tasks. In Figure 9(a), we plot the changes in accuracy for various temperatures τ. One can see that reducing the randomness of DPM-SNC gives a better prediction in real-world node classification tasks.

We also consider sampling various numbers of predictions for node-wise aggregation to improve performance. In Figure 9(b), even the number of samples is increased to 2^{10}, the deterministic and the node-wise aggregation inference schemes show similar accuracy, and there are only minor performance improvements on Citeseer. In practice, we use deterministic inference in the node classification tasks since the node-wise aggregation requires a relatively long time.