Propagate & Distill: Towards Effective Graph Learners Using Propagation-Embracing MLPs

Yong-Min Shin
Yonsei Univsersity
jordan3414@yonsei.ac.kr

Won-Yong Shin*
Yonsei Univsersity
wy.shin@yonsei.ac.kr

Abstract

Recent studies attempted to utilize multilayer perceptrons (MLPs) to solve semi-supervised node classification on graphs, by training a student MLP by knowledge distillation from a teacher graph neural network (GNN). While previous studies have focused mostly on training the student MLP by matching the output probability distributions between the teacher and student models during distillation, it has not been systematically studied how to inject the structural information in an *explicit* and *interpretable* manner. Inspired by GNNs that separate feature transformation T and propagation Π, we re-frame the distillation process as making the student MLP learn both T and Π. Although this can be achieved by applying the inverse propagation Π^{-1} before distillation from the teacher, it still comes with a high computational cost from large matrix multiplications during training. To solve this problem, we propose **Propagate & Distill (P&D)**, which propagates the output of the teacher before distillation, which can be interpreted as an approximate process of the inverse propagation. We demonstrate that **P&D** can readily improve the performance of the student MLP.

1 Introduction

Although message passing built upon the graph structure is crucial to the graph neural network (GNN)'s performance [1, 2], it is known that this also causes a slow inference time [3, 4], which sets a constraint on various real-world applications of GNNs, in particular where fast inference time is essential (*e.g.,* web recommendation [5, 6]). Very recently, GLNN [4] proposed to replace GNNs with multiplayer perceptrons (MLPs), while training the student MLP with knowledge distillation (KD) [7] from a GNN teacher. By this approach, it was shown that the inference time has become over $\times 100$ faster, while resulting in a satisfactory performance of the student MLP in semi-supervised node classification (SSNC). However, in such a scenario of GNN-to-MLP KDs, the student MLP is unable to utilize the structural information of graphs as input, resulting in a large information gap between the teacher GNN and the student MLP. In light of this, the main objective of GNN-to-MLP KDs is to enable the weights of the student MLP to *learn* the graph structure so that, the student MLP achieves the prediction accuracy on par with the teacher GNN.

Several follow-up studies on GNN-to-MLP KDs since the success of GLNN [4] mainly relied a common solution by leveraging the structural information as a part of the input to the student MLP [8–10] to achieve state-of-the-art performance, which however poses the following technical challenges.

- **(Challenge 1)** In [8], the positional embeddings learned by DeepWalk [11] are concatenated into the node features as input to the student MLP. This necessitates consistent re-computation when the underlying graph evolves for maintenance.
- **(Challenge 2**) Other GNN-to-MLP approaches either directly utilize rows of the adjacency matrix, which makes the input dimension of the MLP dependent on the number of nodes [9],

*Corresponding Author.

Shin et al., Propagate & Distill: Towards Effective Graph Learners Using Propagation-Embracing MLPs (Extended Abstract). Presented at the Second Learning on Graphs Conference (LoG 2023), Virtual Event, November 27–30, 2023.

or require a specific design of GNNs that is able to provide structural node features [10], thus reducing the flexibility of the model design.

To tackle the above challenges, we aim to devise a new methodology such that MLPs are capable of learning the structural information while 1) *fixing* the input of the student MLP to the *node features only* and 2) injecting structural information during *training*. Additionally, since most GNN models stack up to only a few layers (*e.g.*, 2 GNN layers) [12, 13] in practice due to the oversmoothing problem [14, 15], depending solely on the teacher GNN's output may disable the student MLP to capture high-order connectivity information, leaving a room for further performance improvement.

To achieve the goal of graph structure learning for MLPs along with the aforementioned constraints, we gain an insight from GNNs that separate feature transformation T and propagation Π [16–19]. Based on these studies, we aim to further boost the performance of the student MLP using Π during the KD process in an *explicit* and *interpretable* manner, which encodes a global structural view of the underlying graph. At its core, this approach begins by regarding the teacher GNN's output as a *base prediction* rather than the final prediction, allowing us to arrive at a formulation where the output of the student MLP first passes through an inverse propagation Π^{-1} before being matched with the teacher GNN's output during KD. Although this approach can be interpreted as training the student MLP in such a way that it behaves as a graph learner embracing the propagation Π by learning both T and Π, it requires large matrix multiplications for each feed-forward process during the training process in KD. As a more efficient workaround, we propose **Propagate & Distill (P&D)**, which approximates Π^{-1} by recursively propagating the teacher GNN's output over the graph. Our approach also allows a room for more flexibility by adopting different propagation rules, akin to prior studies on label propagation [17, 20, 21]. We demonstrate the superiority of **P&D** on popular real-world benchmark datasets, and show that stronger propagation generally leads to better performance.

In summary, our contributions are as follows:

- We present **P&D**, a simple yet effective GNN-to-MLP distillation method that allows additional structural information to be injected during training by recursively propagating the output of the teacher GNN;
- We empirically validate the effectiveness of **P&D** using real-world graph benchmark datasets for both transductive and inductive settings;
- We demonstrate that deeper and stronger propagation in **P&D** generally tends to achieve better performance.

2 Methodology

In this section, we elaborate on our proposed framework **P&D**. We first describe the background of prior work that attempted to separate feature transformation and propagation. Then, we describe our formulation eventually leading to the proposed framework.

Background. Typical GNN models stack multiple message passing layers, each of which consists of the propagation phase and the transformation phase [22]. On the other hand, a handful of prior studies including [16–19] proposed to separate feature transformation and propagation in the GNN model. Given a feature vector $\mathbf{x}_i$, a feature transformation $\mathcal{T} : \mathbf{x}_i \rightarrow \mathbf{h}_i^t$ is first applied to calculate the base prediction $\mathbf{h}_i^t$, and then the GNN model propagates $\mathbf{h}_i^t$ along the underlying graph by a propagation operation $\Pi : \mathbf{h}_i^t \rightarrow \mathbf{p}_i^t$ to get the final prediction $\mathbf{p}_i^t$. As an example, PPNP [16] employed an MLP model to learn the proper feature transformation $\mathcal{T}$ and utilized personalized PageRank (PPR) as the propagation operation Π. In PPNP, the propagation operation $\Pi = \Pi_{\text{PPR}}$ is characterized as $\Pi_{\text{PPR}} = (1-\gamma)(I_{|\mathcal{V}|} - \gamma\tilde{A})^{-1}$, where $1-\gamma \in (0,1]$ is the restart probability, $\mathcal{V}$ is the set of nodes, $I_{|\mathcal{V}|} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is an identity matrix, and $\tilde{A}$ is the symmetrically normalized adjacency matrix with self-loops. Such separation-based approaches have been shown to effectively encode the (global) structural information while avoiding the oversmoothing effect [16, 19]. In our study, to further boost the performance of the student MLP f, we re-frame the GNN-to-MLP KD problem by regarding the teacher GNN as GNN $\approx \Pi' \circ T$, where Π' does not perform enough propagation to benefit our graph learning task. In this new viewpoint, we do not want to solely rely on the output logits P^t of the teacher GNN. Rather, $P^t \approx (\Pi' \circ T)(X)$ can be further enhanced by an additional propagation Π_{PPR} to complement Π', and we set $f(X) = (\Pi_{\text{PPR}} \circ \Pi' \circ T)(X)$ as our ideal objective of KD. However, since $\Pi = \Pi_{\text{PPR}}$ is defined as an (computationally expensive) inverse matrix, it is more reasonable to

consider the distillation loss $\mathrm{KL}((\Pi^{-1} \circ f)(X), (\Pi' \circ T)(X))$, which results in:

$$\mathcal{L}_{\mathrm{KL}} = \mathrm{KL}((\Pi^{-1} \circ f)(X), P^t) = \mathrm{KL}((2I_{|\mathcal{V}|} - \gamma\tilde{A})P^s, P^t), \tag{1}$$

where KL denotes the Kullback–Leibler divergence, P^s is the student MLP's output, and γ comes from Π_{PPR}.[2] In Eq. (1), we include an additional identity matrix, which can be interpreted as an additional skip-connection alongside Π^{-1}. By multiplying the term $2I_{|\mathcal{V}|} - \gamma\tilde{A}$ by P^s before calculating the loss, this formulation explicitly involves the structural information during training.

Propagate & Distill (P&D). A downside of Eq. (1) is that, for every loss calculation during the KD process, we need to constantly multiply $2I_{|\mathcal{V}|} - \gamma\tilde{A}$, thus increasing the computational cost. To remedy this, we choose a computationally efficient alternative formulation of Eq. (1) by *approximating* the inverse matrix calculation in Π_{PPR} along with a recursive formula, similarly as in the approach of APPNP [16]. To this end, we propose **P&D**, which discovers an approximate propagation function $\bar{\Pi} \approx \Pi$, where $\bar{\Pi}$ is defined as a recursive formula that is applied to the output of the *teacher GNN's prediction* instead of applying the inverse propagation function Π^{-1} to the output of the student MLP. In other words, $\bar{\Pi}$ propagates P^t along the underlying graph by recursively applying

$$P^t_{l+1} = \gamma\tilde{A}P^t_l + (1-\gamma)P^t_l, \tag{2}$$

where we initially set $P^t_1 = P^t$ for $l = 1, \cdots, T$; and $\gamma \in (0, 1]$ is a coefficient controlling the propagation strength through neighbors of each node. Denoting the propagation operation in **P&D** as $\bar{\Pi}(P^t, \tilde{A})$, we now formulate our new distillation loss function $\mathcal{L}_{\textbf{P\&D}}$ as follows:

$$\mathcal{L}_{\textbf{P\&D}} = \mathrm{KL}(P^s, \bar{\Pi}(P^t, \tilde{A})), \tag{3}$$

which only requires an additional calculation of $\bar{\Pi}(P^t, \tilde{A})$, leaving the rest of the KD process same as GLNN [4]. Furthermore, this approach not only introduces another natural interpretation, but also allows a room for flexibility in the design of a recursive formula. Precisely, Eq. (2) can be seen as iteratively smoothing the output of the teacher's prediction along the graph structure, which is closely related to classic label propagation (LP) methods [20, 21] that propagate node label information rather than probability vectors. As in LP, **P&D** also takes advantage of the homophily assumption to potentially correct the predictions of incorrectly-predicted nodes with the aid of their (mostly correctly predicted) neighbors.[3] Furthermore, thanks to the flexibility of the LP family, we introduce another variant, named as **P&D**-fix. In this version, the l-th iteration of propagation now becomes

$$\begin{aligned} &\text{(Step 1) } P^t_{l+1} = \gamma\tilde{A}P^t_l + (1-\gamma)P^t_l, \\ &\text{(Step 2) } P^t_{l+1}[j,:] = P^t[j,:] \text{ for } j \in \mathcal{V}_T, \end{aligned} \tag{4}$$

where $\mathcal{V}_T$ denotes the set of training nodes. Different from **P&D**, for every iteration, the output probability of training nodes gets manually replaced by the initial output probability (see Step 2 in Eq. (4)). Adding Step 2 during propagation will lead to the initial output probability for some nodes in the training set as their predictions are expected to be nearly correct. In later descriptions, we denote **P&D** and **P&D**-fix as the versions using functions $\bar{\Pi}$ and $\bar{\Pi}_{\mathrm{fix}}$, respectively. We also denote the previous inverse propagation approach in Eq. (1) as InvKD. Note that, during inference, we use P^s as the student model's prediction, *i.e.,* $f(\mathbf{x}_i) = \mathbf{h}^s_i$.

3 Main Results

In this section, we present experimental results to validate the effectiveness of **P&D** and **P&D**-fix with further empirical analyses of the approximate propagation function $\bar{\Pi}$.[4]

Experimental setup. In our experiments, we mostly follow the settings of prior studies [4, 23]. Specifically, we focus only on the KL divergence loss (without the cross-entropy loss) for all experiments. We adopt a 2-layer GraphSAGE model [24] with 128 hidden dimensions. We use the Adam optimizer [25], batch size of 512, and early stopping with patience 50 during training. We report the average accuracy of the student MLP from 10 trials.

[2]The constant term $(1-\gamma)^{-1}$ in Π^{-1}_{PPR} can be ignored as we normalize both terms in the KL divergence loss.

[3]We refer to Appendix **??** for a theoretical analysis on the role of homophily.

[4]We refer to Appendix G and F for further details.

Table 1: Node classification accuracy (%) for five different datasets in transductive and inductive settings. The columns represent the performance of the teacher GNN model, plain MLP model without distillation, GLNN [4], InvKD, and two versions of **P&D**. For each dataset, the performance of the best method is denoted in bold font.

Transductive	Teacher GNN	Plain MLP	GLNN	InvKD	**P&D**	**P&D**-fix
Cora	78.81 ± 2.00	59.18 ± 1.60	80.73 ± 3.42	82.22 ± 1.45	82.16 ± 1.98	**82.29** ± 1.60
CiteSeer	70.62 ± 2.24	58.51 ± 1.88	71.19 ± 1.36	74.08 ± 1.82	73.38 ± 1.39	**74.93** ± 1.63
Pubmed	75.49 ± 2.25	68.39 ± 3.09	76.39 ± 2.36	77.22 ± 1.98	77.88 ± 2.89	**78.11** ± 2.89
A-Computer	82.69 ± 1.26	67.79 ± 2.16	83.61 ± 1.49	**83.81** ± 1.16	82.06 ± 1.58	83.21 ± 1.21
A-Photo	90.99 ± 1.34	77.29 ± 1.79	92.72 ± 1.11	92.83 ± 1.22	92.91 ± 1.31	**93.02** ± 1.32
Inductive						
Cora	80.61 ± 1.81	59.44 ± 3.36	73.07 ± 1.90	**75.18** ± 1.26	72.27 ± 2.74	71.24 ± 3.45
CiteSeer	69.83 ± 4.16	59.34 ± 4.61	68.37 ± 4.22	71.93 ± 3.16	**72.87** ± 2.57	72.69 ± 2.40
Pubmed	75.25 ± 2.42	68.29 ± 3.26	75.01 ± 2.20	76.49 ± 2.47	76.49 ± 2.47	**76.58** ± 2.34
A-Computer	83.06 ± 1.81	67.86 ± 2.16	79.77 ± 1.72	80.04 ± 1.90	80.28 ± 1.79	**80.38** ± 1.59
A-Photo	91.21 ± 1.10	77.44 ± 1.50	89.73 ± 1.18	**90.28** ± 1.04	90.23 ± 1.02	89.87 ± 1.03

Table 2: Node classification accuracy (%) according to different T's for the Cora dataset. The best performing cases are underlined.

T	≤ 5	10	20	50
Trans.	82.16	82.88 (↑0.72)	<u>83.03</u> (↑0.87)	82.38 (↑0.22)
Ind.	71.59	<u>72.27</u> (↑0.68)	71.31 (↓0.28)	71.85 (↑0.26)

Table 3: Node classification accuracy (%) according to different γ's for the Cora dataset. The performance gain of the case of $\gamma = 0.9$ over the case of $\gamma = 0.1$ is displayed in the parenthesis.

Transductive		*Inductive*	
$\gamma = 0.1$	$\gamma = 0.9$	$\gamma = 0.1$	$\gamma = 0.9$
80.35	82.16 (↑1.81)	70.87	71.99 (↑1.12)

Datasets. We use the Cora, CiteSeer, Pubmed [26, 27], A-Computer, and A-Photo [12] datasets. We choose 20 / 30 nodes per class for the training / validation sets as in [4, 12]. For the inductive setting, we further sample 20% of test nodes to be held out during training.

Scenario settings. In the transductive setting, we use the set of edges $\mathcal{E}$ and node features X for all nodes in $\mathcal{V}$, along with the label information in the set of training nodes $\mathcal{V}_T$. In the inductive setting, all edges that connect nodes in the inductive subset $\mathcal{U}_{\text{ind}}$ and the rest of the graph (i.e., $\mathcal{V}_T \cup \mathcal{U}_{\text{obs}}$) are removed, and remain disconnected during the test phase, following [4].

Experimental results. Table 1 shows the performance comparison with GLNN [4] as well as the teacher GNN and plain MLP models, in terms of the node classification accuracy for all five benchmark datasets. First, we observe that using one of InvKD, **P&D**, and **P&D**-fix consistently outperforms the benchmark methods regardless of datasets and scenario settings. For example, in the transductive setting using the CiteSeer dataset, **P&D**-fix exhibits the best performance with a gain of 3.74% over GLNN, and in the inductive setting using the Cora dataset, InvKD is the best performer while showing a gain of 2.11% over GLNN. Such a benefit from the inductive setting is meaningful since the propagation is performed without access to test nodes.

We also investigate how the total number of iterations T and the propagation strength γ in Eq. (2) affects the performance. Here, we perform the analysis using **P&D** as our main framework.[5] To see how the performance behaves with T, we consider four cases: $T \in \{1, 2, 5\}$, $T = 10$, $T = 20$, and $T = 50$. We measure the performance gain compared to the first case (*i.e.*, $T \in \{1, 2, 5\}$) using the Cora dataset. Table 2 shows that, in both transductive/inductive settings, the best performance can be achieved when T is sufficiently large (*i.e.*, $T \geq 10$). Next, to see how the performance behaves with γ, we consider two cases: $\gamma = 0.1$ and $\gamma = 0.9$. Table 3 shows that stronger propagation (*i.e.*, $\gamma = 0.9$) leads to higher performance in both cases.

4 Conclusion

We presented **P&D**, a simple yet effective method to boost the performance of MLP models trained by distillation from a teacher GNN model. We empirically showed that applying an approximate propagation $\bar{\Pi}$ to the teacher GNN's output eventually benefits the student MLP model after KD on real-world graph benchmark datasets for both transductive and inductive settings. Our future work includes the potential enhancement of the inverse propagation in learning Π^{-1} as a separate model.

[5] We refer to Appendix C for the analysis for other datasets and **P&D**-fix.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1A2C3004345, No. RS-2023-00220762).

References

[1] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proc. 7th Int. Conf. Learn. Representations (ICLR)*, New Orleans, LA, May 2019. 1

[2] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, pages 4602–4609, Honolulu, HI, Jan.–Feb. 2019. 1

[3] Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. TinyGNN: Learning efficient graph neural networks. In *Proc. 26th ACM SIGKDD Conf. Knowl. Discovery Data Mining (KDD)*, pages 1848–1856, Virtual Event, Aug. 2020. 1

[4] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. In *Proc. 10th Int. Conf. Learn. Representations (ICLR)*, Virtual Event, Apr. 2022. 1, 3, 4, 7, 10, 16

[5] Junheng Hao, Tong Zhao, Jin Li, Xin Luna Dong, Christos Faloutsos, Yizhou Sun, and Wei Wang. P-Companion: A principled framework for diversified complementary product recommendation. In *29th ACM Int. Conf. Inf. Knowl. Management (CIKM)*, pages 2517–2524, Virtual Event, Oct. 2020. 1

[6] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. AGL: A scalable system for industrial-purpose graph machine learning. *Proc. VLDB Endow.*, 13(12):3125–3137, 2020. 1

[7] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 7

[8] Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh V. Chawla. Learning MLPs on graphs: A unified view of effectiveness, robustness, and efficiency. In *Proc. 11th Int. Conf. Learn. Representations (ICLR)*, Kigali Rwanda, May 2023. 1, 7, 10

[9] Jie Chen, Shouzhen Chen, Mingyuan Bai, Junbin Gao, Junping Zhang, and Jian Pu. SA-MLP: Distilling graph knowledge from GNNs into structure-aware MLP. *arXiv preprint arXiv:2210.09609*, 2022. 1, 7

[10] Wenqing Zheng, Edward W. Huang, Nikhil Rao, Sumeet Katariya, Zhangyang Wang, and Karthik Subbian. Cold Brew: Distilling graph node representations with incomplete or missing neighborhoods. In *Proc. 10th Int. Conf. Learn. Representations (ICLR)*, Virtual Event, Apr. 2022. 1, 2, 7

[11] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Proc. 20th ACM SIGKDD Conf. Knowl. Discovery Data Mining (KDD)*, pages 701–710, 2014. 1

[12] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. In *Proc. NeurIPS Relational Representation Learning Workshop*, Dec 2018. 2, 4

[13] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *Proc. Int. Conf. Comput. Vision (ICCV)*, pages 9266–9275, Seoul, South Korea, Oct.–Nov. 2019. 2

[14] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 32, 2018. 2

[15] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, pages 3438–3445, New York, NY, Feb. 2020. 2

[16] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proc. 7th Int. Conf. Learn. Representations (ICLR)*, New Orleans, LA, May 2019. 2, 3, 8

[17] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks. In *Proc. 9th Int. Conf. Learn. Representations (ICLR)*, Virtual Event, May 2021. 2

[18] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proc. 26th ACM SIGKDD Conf. Knowl. Discovery Data Mining (KDD)*, pages 2464–2473, Virtual Event, Aug. 2020.

[19] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *Proc. 9th Int. Conf. Learn. Representations (ICLR)*, Virtual Event, May 2021. 2

[20] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Proc. 17th Int. Conf. Neural Inf. Process. Syst. (NeurIPS) 2003*, pages 321–328, Vancouver and Whistler, Canada, Dec. 2003. 2, 3

[21] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proc. 12th Int. Conf. Mach. Learn. (ICML)*, pages 912–919, Washington, DC, Aug. 2003. 2, 3

[22] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, pages 1263–1272, Sydney, Australia, Aug. 2017. 2

[23] Cheng Yang, Jiawei Liu, and Chuan Shi. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *Proc. Web Conf. WWW,*, pages 1227–1237, Virtual Event / Ljubljana, Slovenia, Apr. 2021. 3, 10

[24] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, pages 1024–1034, Dec. 2017. 3

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. 3rd Int. Conf. Learn. Representations (ICLR)*, San Diego, CA, May 2015. 3

[26] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008. 4

[27] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proc. 33th Int. Conf. Mach. Learn. (ICML)*, pages 40–48, New York City, NY, Jun. 2016. 4, 9

[28] Yang Hu, Haoxuan You, Zhecan Wang, Zhicheng Wang, Erjin Zhou, and Yue Gao. Graph-MLP: Node classification without message passing in graph. *arXiv preprint arXiv:2106.04051*, 2021. 7

[29] Wei Dong, Junsheng Wu, Yi Luo, Zongyuan Ge, and Peng Wang. Node representation learning in graph via node-to-neighbourhood mutual information maximization. In *Proc. Conf. Comput. Vision Pattern Recognit. (CVPR)*, pages 16599–16608, New Orleans, LA, Jun. 2022.

[30] Taiqiang Wu, Zhe Zhao, Jiahao Wang, Xingyu Bai, Lei Wang, Ngai Wong, and Yujiu Yang. Edge-free but structure-aware: Prototype-guided knowledge distillation from gnns to mlps. *arXiv preprint arXiv:2303.13763*, 2023.

[31] Lirong Wu, Haitao Lin, Yufei Huang, Tianyu Fan, and Stan Z. Li. Extracting low-/high-frequency knowledge from graph neural networks and injecting it into mlps: An effective gnn-to-mlp distillation framework. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, pages 10351–10360, Washington, DC, Feb. 2023. 7

[32] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, pages 535–541, Philadelphia, PA, Aug. 2006. 7

[33] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819, 2021. 7

[34] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. In *Proc. 34th Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Virtual Event, Dec. 2020. 8

[35] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. In *30th ACM Int. Conf. Inf. Knowl. Management (CIKM)*, pages 2517–2524, Virtual Event, Queensland, Australia, Nov. 2021. 8

[36] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Proc. 34th Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Virtual Event, Dec. 2020. 11

A Preliminaries

In this section, we summarize several preliminaries to our work, along with basic notations.

Semi-supervised node classification. In SSNC, we are given a graph dataset $G = (\mathcal{V}, \mathcal{E}, X)$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $X \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the node feature matrix where the i-th row $\mathbf{x}_i = X[i,:] \in \mathbb{R}^d$ is the d-dimensional feature vector of node $v_i \in \mathcal{V}$. We also denote $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ as the adjacency matrix to represent $\mathcal{E}$, where $A[i,j] = 1$ if $(i,j) \in \mathcal{E}$, and 0 elsewhere. The degree matrix $D = \text{diag}(A\mathbf{1}_{|\mathcal{V}|})$ is a diagonal matrix whose diagonal entry $D[i,i]$ represents the number of neighbors for v_i, where $\mathbf{1}_{|\mathcal{V}|}$ is the all-ones vector of dimension $|\mathcal{V}|$. Alongside G, $\mathcal{Y}$ indicates the set of class labels, and each node v_i is associated with a ground truth label $y_i \in \mathcal{Y}$. Typically, y_i is encoded as a one-hot vector $\mathbf{y}_i \in \mathbb{R}^{|\mathcal{Y}|}$. In SSNC, we assume that only a small subset of nodes $\mathcal{V}_T \subset \mathcal{V}$ have their class labels known during training. The objective of SSNC is to predict the class label for the rest of the nodes in $\mathcal{U} = \mathcal{V} \setminus \mathcal{V}_T$. In the transductive setting, we assume that access to information other than its labels (i.e., node features and associated edges) for all nodes in $\mathcal{U}$ is available during training. In the inductive setting, a held-out subset of nodes in $\mathcal{U}$ by separating into two disjoint subsets, namely the observed subset $\mathcal{U}_{\text{obs}}$ and the inductive subset $\mathcal{U}_{\text{ind}}$ ($\mathcal{U} = \mathcal{U}_{\text{obs}} \cup \mathcal{U}_{\text{ind}}$). The inductive subset $\mathcal{U}_{\text{ind}}$ is completely unknown during training, and the objective is to predict the labels of those unseen nodes.

KD from GNNs to MLPs. Recently, several studies have put their efforts to leverage MLP models as the main architecture for SSNC [4, 8–10, 28–31]. Most of these attempts adopt the KD framework [7, 32, 33] by transferring knowledge from a teacher GNN to a student MLP. As the core component, knowledge transfer is carried out by matching soft labels via a loss function $\mathcal{L}_{\text{KL}}$, which plays a role of matching the output probability distributions between the teacher and student models with respect to the Kullback–Leibler (KL) divergence. Although other distillation designs have been proposed since [7], distillation via $\mathcal{L}_{\text{KL}}$ has been a popular choice and is adopted in lots of follow-up studies that aim to distill knowledge from GNNs to MLPs.

More precisely, in our distillation setting where an MLP is trained via distillation from a teacher GNN, we first assume that the output of the teacher GNN, $H^t \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{Y}|}$, is given, where $\mathbf{h}_i^t = H^t[i,:]$ represents the output logit for node v_i. The objective of KD from GNNs to MLPs is to train the student MLP model f, which returns an output logit $f(\mathbf{x}_i) = \mathbf{h}_i^s$ for a given node feature vector $\mathbf{x}_i$ of node v_i as input. The two output logits $\mathbf{h}_i^t$ and $\mathbf{h}_i^s$ are transformed into class probability distributions by a softmax function, i.e., $\mathbf{p}_i^t = \texttt{softmax}(\mathbf{h}_i^t)$ and $\mathbf{p}_i^s = \texttt{softmax}(\mathbf{h}_i^s)$, respectively. During distillation, $\mathcal{L}_{\text{KL}} \triangleq \text{KL}(\mathbf{p}_i^s, \mathbf{p}_i^t)$ compares the student's output probability $\mathbf{p}_i^s$ and the teacher's output probability $\mathbf{p}_i^t$ by a KL divergence loss. A mix of $\text{KL}(\mathbf{p}_i^s, \mathbf{p}_i^t)$ and the cross-entropy loss, denoted as $\text{CE}(\mathbf{p}_i^s, \mathbf{y}_i)$, with labeled nodes is used as a final loss function:

$$\mathcal{L}_{\text{Distill}} = \alpha \sum_{i \in \mathcal{V}_T} \text{CE}(\mathbf{p}_i^s, \mathbf{y}_i) + (1-\alpha) \sum_{i \in \mathcal{V}} \text{KL}(\mathbf{p}_i^s, \mathbf{p}_i^t), \tag{5}$$

where $\alpha \in [0,1]$ is a mixing parameter. After training, only the MLP model f is used during inference, which dramatically improve the computational efficiency since the feed-forward process basically involves only matrix multiplication and element-wise operations, without message passing [4, 8]. Since a majority of GNN-to-MLP distillation methods adopt only the second term as their distillation loss [4, 8, 9], we also focus on Eq. (5) to set $\alpha = 0$ in the KL divergence loss in our study.

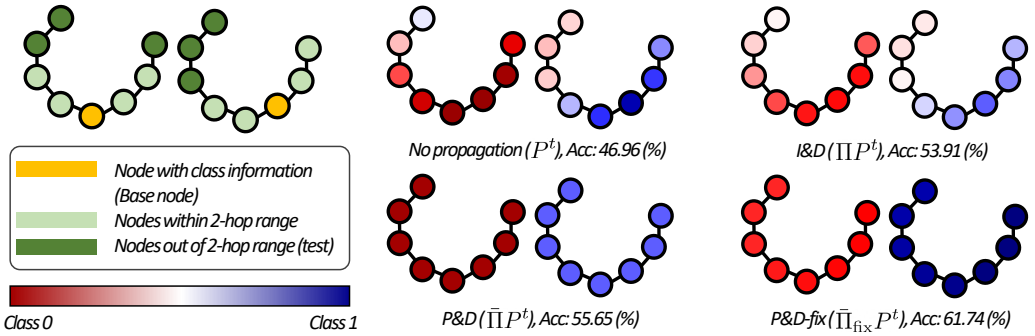


Figure 1: Visualization of the effect of various propagation functions in InvKD and **P&D** on the synthetic Chains dataset.

B Case Study on Interpretations

In order to provide interpretations on the benefits of injecting additional structure information in InvKD and **P&D** during distillation, we perform experiments on the synthetic Chains dataset [34], which consists of 30 chain graphs of a fixed length of 8. Fig. 1 visualizes 2 chains for each propagation for ease of presentation. All nodes in the same chain are assigned to the same class, and the class information is provided as a one-hot representation in the feature vector of *one* of the nodes (the base node) in the chain. To train the teacher GNN, we adopt a 2-layer GraphSAGE model, which is thus able to exploit connectivities only within 2-hop neighbors of the base node. Then, we plot P^t, ΠP^t, $\bar{\Pi} P^t$, and $\bar{\Pi}_{\text{fix}} P^t$, which correspond to the case for GLNN, InvKD, **P&D**, **P&D**-fix, respectively.[6] Compared to P^t where the teacher GNN only correctly predicts the nodes near base node, other propagations ΠP^t, $\bar{\Pi} P^t$, and $\bar{\Pi}_{\text{fix}} P^t$ further spread the correct label information along the graph, while self-correcting the base prediction of P^t. Additionally, we evaluate the accuracy of the student MLP on the nodes further than 2-hops away from the base nodes (see dark green nodes in the left part of Fig. 1). We can observe that self-correction indeed benefits the student MLP. For example, for the nodes out of the 2-hop range, using $\bar{\Pi}_{\text{fix}} P^t$ results in the accuracy of 61.74% compared to the case of using P^t showing the accuracy of 46.96%. This case study clearly validates the effect of our inverse/recursive propagation functions.

C Further Results of Propagation Analysis

In this subsection, we provide additional results of the effects of T and γ for both **P&D** and **P&D**-fix. Table 4 and 5 show the effects of T and γ, respectively, for **P&D** and **P&D**-fix. Table 4 shows that indicating that higher values of T are more beneficial holds for other datasets on both **P&D** and **P&D**-fix, with the exception of **P&D** on Citeseer for the inductive setting. Also, Table 5 indicates that higher values of γ are more beneficial holds for other datasets on both **P&D** and **P&D**-fix.

D Connections to Graph Signal Denoising

We can interpret the basis of our framework as applying graph signal denoising (GSD) to the student MLP. We first present the following theorem that connects propagation and GSD:

Theorem D.1 (GSD of PPNP [35])**.** *Given a noisy signal $S \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{Y}|}$, PPNP [16] solves a GSD problem, where the goal is to recover a clean signal $F \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{Y}|}$ by solving the following optimization problem:*

$$arg\min_{F} \mathcal{L}_{GSD} = ||F - S||^2 + (1/(1-\gamma) - 1) tr(F^\top L F), \tag{6}$$

where $L = I_{|\mathcal{V}|} - \tilde{A}$ is the Laplacian matrix.

By interpreting $F = f(X)$ and $S = P^t$ in Theorem D.1, InvKD can be seen as training the student MLP to fit a given signal P^t with an additional smoothing constraint $(1/(1-\gamma) - 1)\text{tr}(F^\top L F)$. To

[6] Although we do not directly use ΠP^t in the loss in InvKD, we can consider this as the final prediction when the student MLP ideally achieves the zero loss during training.

Table 4: Node classification accuracy (%) according to different T's for three different datasets. The best performing cases are underlined.

T		**P&D** Cora	**P&D** CiteSeer	**P&D** PubMed	**P&D**-fix Cora	**P&D**-fix CiteSeer	**P&D**-fix PubMed
≤ 5	*Trans.*	82.16	73.72	76.68	81.64	74.93	77.14
	Ind.	71.59	<u>72.87</u>	76.49	71.24	72.69	76.39
10	*Trans.*	82.88 (↑0.72)	73.65 (↓0.07)	<u>77.88</u> (↑1.20)	<u>82.35</u> (↑0.71)	74.01 (↓0.92)	77.06 (↓0.08)
	Ind.	<u>72.27</u> (↑0.68)	72.21 (↓0.66)	76.57 (↑0.08)	70.59 (↓0.65)	70.59 (↓2.10)	<u>76.59</u> (↑0.20)
20	*Trans.*	<u>83.03</u> (↑0.87)	<u>73.74</u> (↑0.02)	76.56 (↓0.12)	81.85 (↑0.21)	74.04 (↓0.89)	77.54 (↑0.40)
	Ind.	71.31 (↓0.28)	72.21 (↓0.66)	<u>76.62</u> (↑0.13)	<u>71.85</u> (↑0.61)	71.85 (↓0.84)	76.36 (↓0.03)
50	*Trans.*	82.38 (↑0.22)	73.38 (↓0.34)	77.01 (↑0.33)	82.29 (↑0.65)	<u>74.97</u> (↑0.04)	<u>78.11</u> (↑0.97)
	Ind.	71.85 (↑0.26)	71.77 (↓1.10)	76.48 (↓0.01)	71.66 (↑0.42)	<u>72.76</u> (↑0.07)	76.58 (↑0.19)

Table 5: Node classification accuracy (%) according to different γ's for three different datasets for **P&D** and **P&D**-fix. The performance gain of the case of $\gamma = 0.9$ over the case of $\gamma = 0.1$ is displayed in the parenthesis.

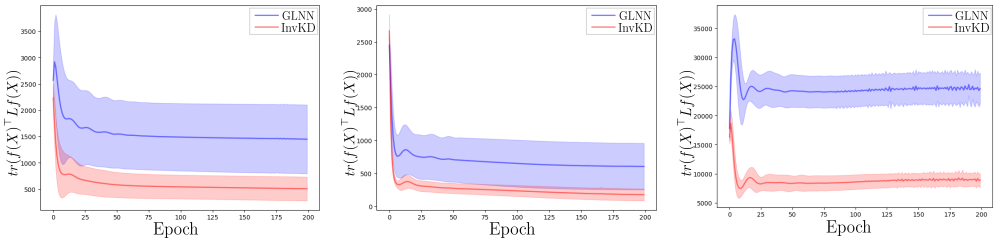
Dataset	**P&D** *Transductive* $\gamma=0.1$	**P&D** *Transductive* $\gamma=0.9$	**P&D** *Inductive* $\gamma=0.1$	**P&D** *Inductive* $\gamma=0.9$	**P&D**-fix *Transductive* $\gamma=0.1$	**P&D**-fix *Transductive* $\gamma=0.9$	**P&D**-fix *Inductive* $\gamma=0.1$	**P&D**-fix *Inductive* $\gamma=0.9$
Cora	80.35	82.16 (↑1.81)	70.87	71.99 (↑1.12)	80.85	82.35 (↑1.50)	69.93	71.24 (↑1.31)
CiteSeer	72.70	73.38 (↑0.68)	71.60	72.87 (↑1.27)	74.47	74.93 (↑0.46)	69.93	72.69 (↑2.76)
PubMed	76.56	77.88 (↑1.32)	75.84	76.49 (↑0.65)	76.89	78.11 (↑1.22)	75.79	76.58 (↑0.79)

explicitly see this effect, we perform a experiment on the Cora, Citeseer, and Pubmed datasets where we plot the regularization term $\text{tr}(f(X)^\top Lf(X))$ versus the number of epochs during training for GLNN and InvKD.

The results in Fig. 2 shows that a much stronger regularization effect for the case of InvKD (red) compared to naïve KD (blue). In conclusion, the inverse propagation Π^{-1} in InvKD further forces the student MLP to return a signal smoothened over the graph.

E Ablation Study on the Inverse Propagation

During our development in InvKD where **P&D** is based on, our inverse propagation function Π^{-1} reveals a form similar to the Laplacian matrix $I_{|\mathcal{V}|} - \tilde{A}$ (see Eq. (1)). Then, a natural question raising is "Is Π^{-1} in InvKD replaceable with alternative operations during distillation?". To answer this question, we run a simple experiment by taking into account two alternatives. First, one can expect that the student model f may also learn the structural information when a convolution function (*i.e.*, the adjacency matrix with self-loops) is applied to the output of the student MLP instead of Π^{-1} since the gradients of the model parameters are also influenced by $\tilde{A}$. Thus, we consider training the student MLP with $\mathcal{L}_{\text{conv}} \triangleq \text{KL}(\tilde{A}P^s, P^t)$. Second, as another alternative, we use $\mathcal{L}_{\text{Distill}}$ in Eq. (5), which does not contain any additional operation on P^s. We follow the same model configurations as those in Section 3, while using the node splits of [27] with full-batch training in the transductive setting.



(a) Cora. (b) CiteSeer. (c) PubMed.

Figure 2: $\mathrm{tr}(f(X)^\top Lf(X))$ versus the number of epochs.

Table 6: Performance comparison when three different loss functions are used during training in the transductive setting.

Dataset	$\mathcal{L}_{\text{conv}}$	$\mathcal{L}_{\text{Distill}}$	$\mathcal{L}_{InvKD}$
Cora	64.78 ± 0.92	73.84 ± 0.52	76.20 ± 0.48
CiteSeer	69.71 ± 0.48	70.23 ± 0.40	72.43 ± 0.56
PubMed	60.27 ± 9.64	76.77 ± 1.02	77.67 ± 0.73

Table 6 shows the experimental results for three different cases when three datasets including Cora, Citeseer, and PubMed are used in the transductive setting. Interestingly, we can observe that the case of using $\mathcal{L}_{\text{conv}}$ exhibits much lower performance than that of other two cases. This is because, while using $\mathcal{L}_{\text{conv}}$ explicitly accommodates the graph structure during distillation, it rather alleviates the pressure for the student MLP to learn the graph structure as $\tilde{A}$ is multiplied anyway during training; however, during inference, the student MLP have lower information on the graph structure with $\tilde{A}$ gone, which eventually harms the performance. Hence, this implies that using a naïve alternative operation may deteriorate the performance and a judicious design of the propagation operation is essential in guaranteeing state-of-the-art performance.

F Statistics of Datasets

We show the table that contains the statistics of the six real-world datasets used in the experiments in Table 7.

Table 7: Statistics of six real-world datasets. NN, NE, NF, and NC denote the number of nodes, the number of edges, the number of node features, the number of classes, respectively.

Dataset	NN	NE	NF	NC
Cora	2,485	5,069	1,433	7
CiteSeer	2,120	3,679	3,703	6
Pubmed	19,717	44,324	500	3
A-computer	13,381	245,778	767	10
A-photo	7,487	119,043	745	8

G Further Experiment Details

G.1 Model Hyperparameters

The hyperparameters for the GraphSAGE model used as the teacher GNN including the number of layers, hidden dimension, learning rate, *etc.* are configured by essentially following the values of prior studies [4, 8, 23]. The full details of the settings are shown in Table 8.

For the student MLP in P&D, we perform a hyperparameter search of the learning rate in [0.01, 0.005, 0.001], weight decay in [0.005, 0.001, 0], dropout ratio in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8], and γ in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]. For P&D and P&D-fix, we perform the same

Table 8: Hyperparameter settings for the teacher GNN used in the experiments.

	Num. of layers	Hidden dim.	Learning rate	Dropout ratio	Weight decay	Fan out
Setting	2	128	0.01	0	0.0005	5,5

hyperparameter search for learning rate, weight decay, and dropout ratio as in P&D, while performing the search of γ in [0.1, 0.9] and T in [1, 2, 5, 10, 20, 50].

G.2 Implementation Details

For implementing our experiments, we use Pytorch version 1.12.0, and all GNN models are implemented using Pytorch Geometric version 2.0.4 and DGL version 0.9.1.

H Theoretical Analysis for Self-Correction via Propagation

In this section, we provide a theoretical analysis of propagating the teacher's output P^t along the graph in the context of *self-correction*. **P&D** relies on the assumption that the underlying graph has high levels of homophily, which measures the ratio of edges where the two connected nodes have the same class label [36]. In this setting, we are interested in analyzing the condition where the prediction of a (incorrectly-predicted) node becomes corrected after one iteration of propagation by Eq. (2). We start by formally addressing basic settings and assumptions, which essentially follow those of [36]. Let us assume that the underlying graph G is regular (*i.e.*, all nodes have a degree of d) and $h \in [0, 1]$ portion of neighbors have the same label for all nodes in $v \in \mathcal{V}$. For each node, the teacher GNN is assumed to be assigned an output probability vector having a probability $p \in [0, 1]$ (with $p > 1/|\mathcal{Y}|$) for the true class label and another probability $(1 - p)/(|\mathcal{Y}| - 1)$ for the rest of the classes if the teacher GNN always makes predictions correctly.

Now, without loss of generality, let us assume that the teacher GNN makes an *incorrect* prediction for a particular node of interest v_* with class 0 as its ground truth label by assigning a new output probability vector

$$P^t[*, :] = \left[q, \frac{1-q}{|\mathcal{Y}|-1}, \cdots, \frac{1-q}{|\mathcal{Y}|-1}\right] \tag{7}$$

with $0 < q < 1/|\mathcal{Y}|$, thus no longer assigning class 0 as its prediction. Additionally, for the *rest* of the nodes, we introduce an error ratio $\epsilon \in (0, 1)$ to the teacher GNN's predictions, which assumes that the teacher model provides incorrect predictions for $\epsilon(|\mathcal{V}| - 1)$ nodes (excluding v_* itself). For the sake of simplicity, we assume that the probability of a node being incorrect by the teacher GNN is independent of its ground truth class label, which establishes the following theorem:

Theorem H.1. *Suppose that the teacher GNN provides incorrect predictions for $\epsilon(|\mathcal{V}| - 1)$ nodes other than node v_* where $\epsilon \in (0, 1)$. Then, using one iteration of propagation in Eq. (2), the prediction of node v_* gets corrected if*

$$q \in \left[\max\left(0, \frac{1}{|\mathcal{Y}|} - \frac{\gamma}{1-\gamma}\left(C - b(\epsilon)\right)\right), \frac{1}{|\mathcal{Y}|}\right], \tag{8}$$

where q is the output probability of v_* corresponding to class 0, C is approximately $\left(1 + \frac{1}{|\mathcal{Y}|}\right) hp - \frac{h+p}{|\mathcal{Y}|}$, γ is the propagation strength in Eq. (2), and $b(\epsilon) = \left(C + \frac{hp}{|\mathcal{Y}|}\right)\epsilon$.

Proof. Before proving Theorem H.1, we first show preliminary calculations that are needed before we proceed with the main proof. Here, we start with the simplest case. Following [36], we calculate the result when one-hot labels are being propagated, which will be used for later calculations. First, without loss of generality, we reorder the label matrix $Y \in \{0, 1\}^{|V| \times |\mathcal{Y}|}$ (and also the rows and

columns of adjacency matrix A) as follows:

$$Y = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{9}$$

We then calculate $(A + I)Y$, which we will progressively modify to $\bar{\Pi}$ during the rest of this section. We take advantage of the neighbor assumptions, which results in:

$$(A+I)Y = \begin{bmatrix} hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\ \vdots & \vdots & \ddots & \vdots \\ hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\ \frac{1-h}{|\mathcal{Y}|-1}d & hd+1 & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1-h}{|\mathcal{Y}|-1}d & hd+1 & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\ \vdots & \vdots & & \vdots \\ \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & hd+1 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & hd+1 \end{bmatrix}. \tag{10}$$

We now start to modify this result of calculating $(A+I)Y$ that more resembles Eq. (2), except that we are still propagating one-hot labels. First, we introduce γ in Eq. (10) and calculate $(\gamma A+(1-\gamma)I)Y$:

$$(\gamma A+(1-\gamma)I)Y = \begin{bmatrix} \gamma hd+(1-\gamma) & \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1}d \\ \vdots & \vdots & \ddots & \vdots \\ \gamma hd+(1-\gamma) & \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1}d \\ \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \gamma hd+(1-\gamma) & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1}d \\ \vdots & \vdots & \ddots & \vdots \\ \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \gamma hd+(1-\gamma) & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1}d \\ \vdots & \vdots & & \vdots \\ \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \gamma hd+(1-\gamma) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \gamma\frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \gamma hd+(1-\gamma) \end{bmatrix}. \tag{11}$$

Finally, we replace A with $\tilde{A} = D^{-1/2}AD^{-1/2}$. Since each node has the same degree d, each signal is now multiplied with $(1/\sqrt{d})^2 = 1/d$ during propagation, eventually cancelling out the d's:

$$(\gamma\tilde{A} + (1-\gamma)I)Y = \begin{bmatrix} \gamma h + (1-\gamma) & \gamma\frac{1-h}{|\mathcal{Y}|-1} & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma h + (1-\gamma) & \gamma\frac{1-h}{|\mathcal{Y}|-1} & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1} \\ \gamma\frac{1-h}{|\mathcal{Y}|-1} & \gamma h + (1-\gamma) & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma\frac{1-h}{|\mathcal{Y}|-1} & \gamma h + (1-\gamma) & \cdots & \gamma\frac{1-h}{|\mathcal{Y}|-1} \\ \vdots & \vdots & & \vdots \\ \gamma\frac{1-h}{|\mathcal{Y}|-1} & \gamma\frac{1-h}{|\mathcal{Y}|-1} & \cdots & \gamma h + (1-\gamma) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma\frac{1-h}{|\mathcal{Y}|-1} & \gamma\frac{1-h}{|\mathcal{Y}|-1} & \cdots & \gamma h + (1-\gamma) \end{bmatrix}. \tag{12}$$

Now, we are ready to change Y to a matrix of probability vectors (*i.e.*, P^t), which finally results in calculating one iteration of $\bar{\Pi}$. In our analysis, we replace Y with a $P^t \in [0,1]^{|V|\times|\mathcal{Y}|}$, where the correct label is predicted with probability p and the rest of the probabilities $(1-p)$ is distributed uniformly for the rest of the classes:

$$P^t = \begin{bmatrix} p & \frac{1-p}{|\mathcal{Y}|-1} & \cdots & \frac{1-p}{|\mathcal{Y}|-1} \\ \vdots & \vdots & \ddots & \vdots \\ p & \frac{1-p}{|\mathcal{Y}|-1} & \cdots & \frac{1-p}{|\mathcal{Y}|-1} \\ \frac{1-p}{|\mathcal{Y}|-1} & p & \cdots & \frac{1-p}{|\mathcal{Y}|-1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1-p}{|\mathcal{Y}|-1} & p & \cdots & \frac{1-p}{|\mathcal{Y}|-1} \\ \vdots & \vdots & & \vdots \\ \frac{1-p}{|\mathcal{Y}|-1} & \frac{1-p}{|\mathcal{Y}|-1} & \cdots & p \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1-p}{|\mathcal{Y}|-1} & \frac{1-p}{|\mathcal{Y}|-1} & \cdots & p \end{bmatrix}. \tag{13}$$

Assuming $1/|\mathcal{Y}| < p \leq 1$ implies that the teacher GNN has the accuracy of 1 (*i.e.*, perfect prediction). Note that setting $p = 1$ reverts P^t to Y. Now, calculating for $(\gamma\tilde{A} + (1-\gamma)I)P^t$ results in:

$$(\gamma\tilde{A} + (1-\gamma)I)P^t = \begin{bmatrix} \beta & \beta' & \cdots & \beta' \\ \vdots & \vdots & \ddots & \vdots \\ \beta & \beta' & \cdots & \beta' \\ \beta' & \beta & \cdots & \beta' \\ \vdots & \vdots & \ddots & \vdots \\ \beta' & \beta & \cdots & \beta' \\ \vdots & \vdots & & \vdots \\ \beta' & \beta' & \cdots & \beta \\ \vdots & \vdots & \ddots & \vdots \\ \beta' & \beta' & \cdots & \beta \end{bmatrix},$$

where

$$\beta = (1-\gamma)p + \gamma hp + \gamma\frac{1-h}{|\mathcal{Y}|-1}(1-p) \tag{14}$$

$$\beta' = (1-\gamma)\frac{1-p}{|\mathcal{Y}|-1} + \gamma\frac{h}{|\mathcal{Y}|-1}(1-p) + \gamma\frac{1-h}{|\mathcal{Y}|-1}p + \gamma(1-h)\frac{|\mathcal{Y}|-2}{(|\mathcal{Y}|-1)^2}(1-p). \tag{15}$$

Now, we are ready to prove Theorem H.1.

In order to prove the theorem, we basically follow the same calculation steps by recalculating the interval for $v_* = v_1$ while including ϵ, which will require more careful considerations. We start with revisiting Eqs. (14) and (15). In Eq. (14), there are three terms, *i.e.*, γhp, $(1-\gamma)q$, and $\gamma(1-p)\frac{1-h}{|\mathcal{Y}|-1}$. For ease of notations, we denote the set of nodes with the same labels with v_1 as S and the rest of neighbors as S' in the previous setting where we assumed that nodes other than v_1 were all correct.

Starting with $(1-\gamma)q$, this term calculates the effect of self-propagation and therefore remains unchanged in the new setting. The term γhp calculates the influence that is aggregated from nodes in S. In the new setting, only $(1-\epsilon)hd$ nodes propagate the probability p (Note that $|S| = hd$), and therefore γhp is changed into $\gamma h\left((1-\epsilon)p + \epsilon\frac{1-p}{|\mathcal{Y}|-1}\right)$. Next, the term $\gamma(1-h)\frac{1-p}{|\mathcal{Y}|-1}$ calculates the influence aggregated from nodes in S', which previously all propagated $\frac{1-p}{|\mathcal{Y}|-1}$. When $\epsilon = 0$, the number of these nodes is $|S'| = (1-h)d$, and in the new setting, $\epsilon(1-h)d$ has their predictions changed, where $\frac{1}{|\mathcal{Y}|-1}$ of them now propagates p. Therefore, this term is now changed into $\gamma(1-h)\left(\epsilon\frac{1}{|\mathcal{Y}|-1}p + \frac{|\mathcal{Y}|-1-\epsilon}{(|\mathcal{Y}|-1)^2}(1-p)\right)$. In summary, in the new setting, β_q becomes

$$\begin{aligned}\beta_{q,\epsilon} = (1-\gamma)q + \gamma h\left((1-\epsilon)p + \epsilon\frac{1-p}{|\mathcal{Y}|-1}\right) \\ + \gamma(1-h)\left(\epsilon\frac{1}{|\mathcal{Y}|-1}p + \frac{|\mathcal{Y}|-1-\epsilon}{(|\mathcal{Y}|-1)^2}(1-p)\right).\end{aligned} \tag{16}$$

In Eq. (15), there are four terms, *i.e.*, $(1-\gamma)\frac{1-q}{|\mathcal{Y}|-1}$, $\gamma\frac{h}{|\mathcal{Y}|-1}(1-p)$, $\gamma\frac{1-h}{|\mathcal{Y}|-1}p$, and $\gamma(1-h)\frac{|\mathcal{Y}|-2}{(|\mathcal{Y}|-1)^2}(1-p)$. Using the assumption that the predictions are uniformly distributed among classes for nodes in S'; without loss of generality, let us calculate the probability regarding the second class label.

Similarly as in β_q, the term $(1-\gamma)\frac{1-q}{|\mathcal{Y}|-1}$ remains unaffected in the new setting as it is the result of self-propagation. The term $\gamma\frac{h}{|\mathcal{Y}|-1}(1-p)$ calculates the influence from nodes that were in S. For these $|S| = hd$ nodes, they previously propagated $\frac{1-p}{|\mathcal{Y}|-1}$. In the new setting, $\epsilon\frac{1}{|\mathcal{Y}|-1}hd$ nodes now predict the second class and propagate p, while the rest of the $\frac{|\mathcal{Y}|-1-\epsilon}{|\mathcal{Y}|-1}hd$ nodes still propagates $\frac{1-p}{|\mathcal{Y}|-1}$. In total, this term is now modified into $\gamma h\left(\frac{\epsilon}{|\mathcal{Y}|-1}p + \frac{|\mathcal{Y}|-1-\epsilon}{(|\mathcal{Y}|-1)^2}(1-p)\right)$. The term $\gamma\frac{1-h}{|\mathcal{Y}|-1}p$ calculates the influence from nodes that previously predicted the second class in S'. The nodes are now split into two groups with ratio $\epsilon : (1-\epsilon)$, where the first group now propagates $\frac{1-p}{|\mathcal{Y}|-1}$ and the latter still propagates p. In total, this term is now modified into $\gamma\frac{1-h}{|\mathcal{Y}|-1}\left(\epsilon\frac{1}{|\mathcal{Y}|-1}(1-p) + (1-\epsilon)p\right)$. Next, the term $\gamma(1-h)(1-p)\frac{|\mathcal{Y}|-2}{(|\mathcal{Y}|-1)^2}$ calculates the influence from nodes that previously did not predict as the second class in S'. Similarly as before, the nodes are now split into two groups with ratio $\frac{1}{|\mathcal{Y}|-1}\epsilon : \frac{|\mathcal{Y}|-1-\epsilon}{|\mathcal{Y}|-1}$, where the first group now (incorrectly) predicts the second class and thus propagates p, while the latter still propagates $\frac{1-p}{|\mathcal{Y}|-1}$.

In total, this term is now modified into $\gamma(1-h)\frac{|\mathcal{Y}|-2}{|\mathcal{Y}|-1}\left(\frac{\epsilon}{|\mathcal{Y}|-1}p+\frac{|\mathcal{Y}|-1-\epsilon}{(|\mathcal{Y}|-1)^2}(1-p)\right)$. In summary, in the new setting, β'_q becomes

$$\begin{aligned}\beta'_{q,\epsilon} &= (1-\gamma)\frac{1-q}{|\mathcal{Y}|-1}+\gamma h\left(\frac{\epsilon}{|\mathcal{Y}|-1}p+\frac{|\mathcal{Y}|-1-\epsilon}{(|\mathcal{Y}|-1)^2}(1-p)\right)\\ &\quad+\gamma\frac{1-h}{|\mathcal{Y}|-1}\left(\epsilon\frac{1}{|\mathcal{Y}|-1}(1-p)+(1-\epsilon)p\right)\\ &\quad+\gamma(1-h)\frac{|\mathcal{Y}|-2}{|\mathcal{Y}|-1}\left(\frac{\epsilon}{|\mathcal{Y}|-1}p+\frac{|\mathcal{Y}|-1-\epsilon}{(|\mathcal{Y}|-1)^2}(1-p)\right).\end{aligned} \tag{17}$$

We can verify that both $\beta_{q,\epsilon=0}$ and $\beta'_{q,\epsilon=0}$ reduce to β_q and β'_q, respectively. Now, in the scenario where the node prediction is corrected after propagation, we need $\beta_{q,\epsilon}>\beta'_{q,\epsilon}$. After calculation with similar approximations when we calculated Eq. (23), we arrive at:

$$\begin{aligned}q &> \frac{1}{|\mathcal{Y}|}-\frac{\gamma}{1-\gamma}\left(\left((1-\epsilon)+\frac{1-2\epsilon}{|\mathcal{Y}|}\right)hp-(1-\epsilon)\frac{h+p}{|\mathcal{Y}|}\right)\\ &= \frac{1}{|\mathcal{Y}|}-\frac{\gamma}{1-\gamma}\left(C-\epsilon\left(C+\frac{hp}{|\mathcal{Y}|}\right)\right),\end{aligned} \tag{18}$$

which concludes the proof of Theorem H.1. □

From Theorem H.1, we can see that an increase of the error ϵ reduces the range of q, enabling the prediction of nodes to get corrected, which means that incorrect predictions from the teacher GNN introduce a more unforgiving environment for self-correction. Moreover, it is worth noting that the acceptable amount of error such that corrections via propagation are possible is upper-bounded by

$$\epsilon < \frac{|\mathcal{Y}|h-1}{(|\mathcal{Y}|+1)h-1}, \tag{19}$$

which monotonically increases with $h\in(1/|\mathcal{Y}|,1]$. This implies that stronger homophily of the underlying graph will result in more tolerance of the prediction error from the teacher GNN.

H.1 Analysis for $\epsilon=0$

Let us also consider a simpler scenario where the teacher GNN makes an incorrect prediction *only* for a single node v_* (*i.e.*, $\epsilon=0$) with class 0 as its ground truth label by assigning a new output probability vector $P^t[*,:]=[q,\frac{1-q}{|\mathcal{Y}-1|},\cdots,\frac{1-q}{|\mathcal{Y}-1|}]$ $(0\le q<1/|\mathcal{Y}|)$, same as in our previous setting in Theorem H.1. In this setting, we establish the following corollary.

Corollary H.2. *Suppose that the teacher make an incorrect prediction only for a single node v_*. Using one iteration of propagation in Eq. (2), the prediction of node v_* gets corrected if*

$$q\in\left[\max\left(0,\frac{1}{|\mathcal{Y}|}-\frac{\gamma}{1-\gamma}C\right),\frac{1}{|\mathcal{Y}|}\right], \tag{20}$$

where q is the output probability of v_* corresponding to class 0 and C is approximately $\left(1+\frac{1}{|\mathcal{Y}|}\right)hp-\frac{h+p}{|\mathcal{Y}|}$.

Proof. Since we consider the case where $\epsilon=0$ (*i.e.*, the teacher GNN provides correct predictions for nodes other than v^*), the resulting vector for the first row of $(\gamma\tilde{A}+(1-\gamma)I)P^t$ can be expressed as:

$$((\gamma\tilde{A}+(1-\gamma)I)P^t)[1,:]=[\beta_q,\underbrace{\beta'_q,\cdots,\beta'_q}_{(|\mathcal{Y}|-1)}]. \tag{21}$$

Intuitively, β_q represents the result after propagation for the correct class, and β'_q represents the rest of the (incorrect) classes. For the incorrect prediction to be corrected after propagation, it requires

$\beta_q > \beta'_q$:

$$(1-\gamma)\left(q - \frac{1-q}{|\mathcal{Y}|-1}\right) > -\gamma hp - \gamma\frac{1-h}{|\mathcal{Y}|-1}(1-p) + \gamma\frac{h}{|\mathcal{Y}|-1}(1-p) + \gamma\frac{1-h}{|\mathcal{Y}|-1}p + \gamma(1-h)\frac{|\mathcal{Y}|-2}{(|\mathcal{Y}|-1)^2}(1-p). \tag{22}$$

Calculation of Eq. (22) can directly reveal the condition for the correction scenario. We can directly derive an interval for q by approximating $\frac{|\mathcal{Y}|-2}{|\mathcal{Y}|-1} \approx 1$, which further reduces Eq. (22) to

$$q > \frac{1}{|\mathcal{Y}|} - \frac{\gamma}{1-\gamma}\left(\left(1+\frac{1}{|\mathcal{Y}|}\right)hp - \frac{h+p}{|\mathcal{Y}|}\right). \tag{23}$$

Denoting $C = \left(1+\frac{1}{|\mathcal{Y}|}\right)hp - \frac{h+p}{|\mathcal{Y}|}$ results in the interval Eq. (22), which concludes the proof of Corollary H.2. □

I Experimental Results on a Larger Dataset

We perform an additional experiment on the large-scale OGB-arxiv dataset, where the number of nodes, number of edges, number of node features, and number of classes are 169,343, 1,166,243, 128, and 40, respectively. The experiment is performed in the transductive scenario for **P&D** and **P&D**-fix, since InvKD is computationally impractical to perform on the large dataset.

Table 9: Node classification accuracy (%) for OGB-arxiv in transductive setting. The columns represent the performance of the teacher GNN model, plain MLP model without distillation, GLNN [4], and two versions of **P&D**. The performance of the best method is denoted in bold font.

Dataset	Teacher GNN	Plain MLP	GLNN	**P&D**	**P&D**-fix
OGB-arxiv	70.64 ± 0.41	55.33 ± 1.54	63.02 ± 0.41	**65.20** ± 0.45	65.14 ± 0.35

As shown in Table 9, experimental results demonstrate that a similar trend is also found for the OGB-arxiv dataset, where **P&**D and **P&D**-fix achieve gains of 2.18% and 2.12% compared to GLNN, respectively.

J Experimental Results on an Alternative Teacher GNN

We perform an additional set of experiments in the transductive setting where APPNP is adopted as the teacher GNN model in addition to GraphSAGE.

Table 10: Node classification accuracy (%) for Cora, Citeseer, Pubmed in the transductive setting employing GraphSAGE and APPNP as the teacher GNN. The columns represent the model used for another teacher GNN, datasets, the performance of GLNN [4], InvKD, and two versions of **P&D**, alongside the performance increase with respect to GLNN in the parenthesis. The case with higher performance increase is denoted as bold font.

Teacher GNN	Dataset	GLNN	InvKD	**P&D**	**P&D**-fix
SAGE	Cora	80.73	82.22 (↑**1.49**)	82.16 (↑**1.43**)	82.29 (↑**1.56**)
	Citeseer	71.19	74.08 (↑**2.89**)	73.38 (↑**2.19**)	74.93 (↑**3.74**)
	Pubmed	76.39	77.22 (↑0.83)	77.88 (↑**1.49**)	78.11 (↑**1.72**)
APPNP	Cora	82.81	83.27 (↑0.46)	83.35 (↑0.54)	83.82 (↑1.01)
	Citeseer	73.02	73.55 (↑0.53)	74.32 (↑0.30)	74.18 (↑1.16)
	Pubmed	75.92	77.24 (↑**1.32**)	77.37 (↑1.45)	77.60 (↑1.68)

Table 10 summarizes the experimental results, including the relative performance gain over GLNN between two cases: using GraphSAGE and APPNP as the teacher GNNs.

From the experimental results, we would like to make the following observations.

- The proposed framework still outperforms GLNN when APPNP is used as the teacher GNN.
- The gain over GLNN tends to be decreased in comparison with the case where GraphSAGE is used as the teacher GNN. This is because a significant portion of the additional structural information that our framework injects is already learned by APPNP due to the similarity in terms of propagation, which potentially makes GLNN more potent.

We expect that developing a more sophisticated propagation method that can combine structural information that cannot be captured by propagation will alleviate this phenomenon, which we aim to tackle in our future work.