PRE-TRAINING PURE GNNS AS GRAPH LEARNERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graphs from different datasets exhibit diverse numbers of features and labels, where each feature or label is associated with different semantic meanings. Such diversity poses challenges in adapting pre-trained graph neural networks (GNNs) to different datasets with a single set of input and output (I/O) module parameters. This raises a fascinating question: Can pure GNNs be pre-trained on diverse datasets, adapting to various datasets effectively without additional effort? To explore this, we propose unified I/O modules that enable pre-training with pure GNNs. Unlike traditional methods that tightly couple parameters to specific datasets, our approach decouples parameters through a shared relation function for the input and uniformly sampled points for the output. These designs effectively resolve the challenges in quantity inconsistency and semantic discrepancies of dataset features and labels. By integrating our I/O modules with various GNN architectures, we demonstrate that pure GNNs can be effective graph learners for direct adaptation to downstream tasks. Pre-training experiments under different setups show that increasing hidden dimensions and the average number of nodes per training dataset enhances model performance. Moreover, fine-tuning the I/O modules with frozen pre-trained graph operators significantly simplifies the model hyperparameter tuning process, achieving superior or comparable performance to supervised models on downstream datasets.

1 INTRODUCTION

Pre-trained foundation models have shown exceptional adaptability across different datasets, such as large language models (LLMs) (OpenAI et al., 2024) for MMLU (Hendrycks et al., 2020) and HumanEval (Chen et al., 2021), large vision models (LVMs) (Kirillov et al., 2023) for CityScapes (Cordts et al., 2016) and PIDRay (Zhang et al., 2023a). A key factor underpinning this capability lies in the unified feature and label space across datasets. In the feature space, although the encoded information varies, the semantic meanings of features remain consistent. For instance, a landscape image and a colored X-ray image can both be represented as 3D tensors, with features corresponding to their RGB values. In the label space, data can be assigned to a limited number of labels.

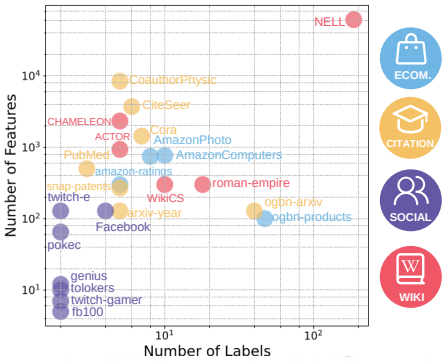


Figure 1: **Diversity of Features and Labels across Graph Datasets.** Graphs from different datasets exhibiting diverse numbers of features and labels, while each feature or label is associated with different semantics.

In contrast to data with the unified feature and label space, graph data typically exhibits diverse numbers of features and labels, with each associated with different semantics (Fig. 1). For instance, feature and label semantics in tolokers (Platonov et al., 2023) correspond to user profiles and user status, whereas those in CoauthorCS (Shchur et al., 2019) correspond to publication keywords and research interest. This inherent diversity presents substantial challenges in developing pre-trained graph models. First, the inconsistent number of features and labels hinders the unified design of the input/output modules (I/O). Second, the semantic discrepancies across graphs impede the effective adaptation of pre-trained models to diverse graph datasets. Traditional I/O fails to tackle these challenges by tightly coupling the quantity and values of the learnable parameters to specific datasets, thereby limiting their adaptability.

To address this problem, methods have proposed leveraging language models (LMs) as I/O for graph neural networks (GNNs) (Liu et al., 2023a; Li et al., 2024b). By transforming graph features and labels into natural language, GNNs integrated with LMs can effectively handle graphs with diverse features and labels. Beyond advancements in pre-training GNNs with LMs, researchers have also explored fine-tuning strategies (Sun et al., 2022; 2023; Huang et al., 2024) to accommodate different features and labels. However, these methods still rely on few-shot knowledge for effective adaptation. It remains an open problem that *can pure GNNs be pre-trained on diverse datasets and directly adapted to downstream datasets?*

In this paper, we propose unified I/O modules to achieve pre-training with pure GNNs by decoupling model parameters from specific datasets. Instead of treating the whole mapping matrix as learnable parameters, unified I/O decomposes the mapping into two successive mapping matrices: one associated with the datasets and the other associated with the hidden space. Our objective is to model the dataset-associated mapping in a transferable manner. For input features, unified I/O employs a shared parametric relation function to learn a predefined number of relations between feature dimensions. The shared function targets and analyzes the same relation patterns across different datasets, which can be employed to construct the dataset-associated mapping with the same set of parameters. For output labels, the module samples uniformly distributed points as pseudo labels, enabling prediction over diverse label spaces. When label information is available for downstream graphs, these uniform points can be aligned with the real labels without additional training.

Our unified I/O modules enable graph pre-training with pure GNNs. Empirical results with different GNN architectures demonstrate that pre-trained pure GNNs can be effective graph learners for direct adaptation to downstream tasks. Specifically, we evaluate the performance of pre-trained models with scaling parameters, scaling training data, and varying domain gaps. Results show that increasing either hidden dimensions or the average number of nodes per graph during pre-training enhances the performance. Moreover, fine-tuning the I/O modules with frozen pre-trained graph operators on downstream graphs substantially reduces the need for extensive hyperparameter tuning, achieving superior or comparable performance with the supervised models. Our contribution can be summarized as

- We propose a novel method to unify the I/O modules for pre-training with general pure GNNs, providing graph operators to simplify the extensive hyperparameter tuning process.
- We demonstrate that pre-trained pure GNNs can serve as effective graph learners on eight classic GNN architectures across diverse real-world datasets.
- We experimentally verify the adaptation performance of pre-trained GNNs with scaling parameters, scaling training data, and different domain gaps.

2 Related Work

Due to space limitations, we provide a brief overview of related work, with a comprehensive discussion in Appendix B. Existing methods for adapting pre-trained graph models fall into two categories: I/O unification and fine-tuning. The former mainly leverage LMs to encode textual attributes for unifying diverse features and labels (Liu et al., 2023a; Wang et al., 2023; Chen et al., 2024a; Kong et al., 2024; Li et al., 2024b; Tang et al., 2024; Zhu et al., 2025). Except for LM-based methods, both parametric (Jing et al., 2023; Zhao et al., 2024b) and parameter-free (Sun et al., 2023; Tang et al., 2024; Sun et al., 2025) unification strategies for pure GNNs have been explored, but they fail to unify label spaces. GraphAny (Zhao et al., 2024a) extends this direction to label unification with linear GNNs. Yet these methods remain limited in scope and often require observed labels, leaving pre-training with general pure GNN architectures an open challenge. The latter fine-tuning methods adapt pre-trained models through graph adapters (Li et al., 2024a; Gui et al., 2024) or graph prompts (Sun et al., 2023; Yu et al., 2025). Distinct from both, our method can be applied to general GNN architectures and enables pre-training for direct adaptation to diverse datasets.

3 Unified I/O for General Pure GNN Architectures

To enable training-free adaptation of pure GNNs, we formulate the unification of the I/O modules as the modeling of the semantics associated with each feature and label (Fig. 2). Specifically, our unified

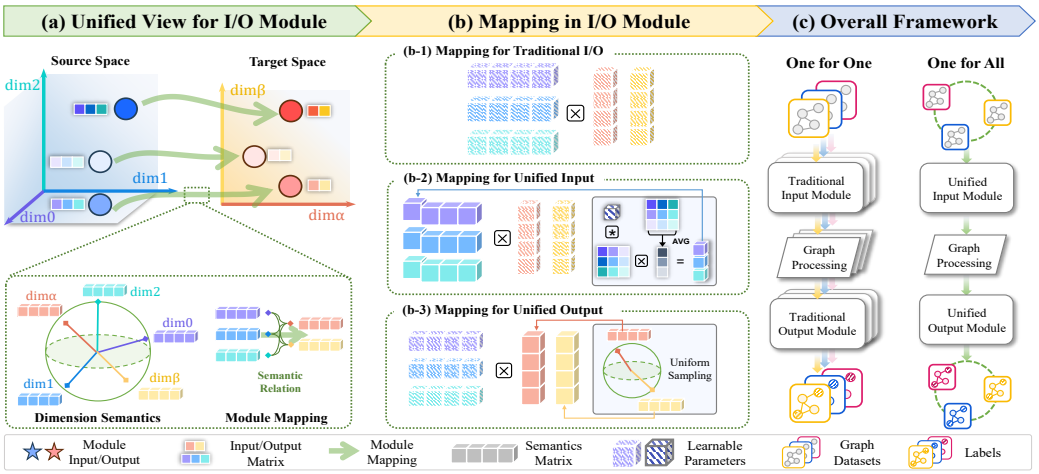


Figure 2: **Illustration of the Unified Input/Output Pipeline.** (a) A linear mapping projects source dimensions onto a set of target dimensions. Each dimension corresponds to certain semantic descriptions that can be embedded in the semantic space. Modeling semantic relations between source and target dimensions results in module mappings. (b) Instead of treating both semantic matrices as parameters, unified I/O modules model the feature semantics as parametric relations and sample label semantics uniformly in the semantic space. (c) The parameter quantity and values in GNNs with traditional I/O are coupled with specific datasets. Our unified I/O decouples the parameters from specific datasets, enabling one set of parameters for different datasets.

input module encodes the feature semantics as parametric relations of the input features, while our output module represents diverse label semantics as uniformly sampled points in the semantic space.

3.1 Problem Setup

Notations. Given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{V} = \{v_1, \cdots, v_n\}$ denotes a set of n nodes and $\mathcal{E} = \{e_{i,j} | v_j \in \mathcal{N}(v_i)\}$ denotes a set of m edges. $\mathcal{N}(\cdot)$ denotes the set of one-hop neighbors for a given node. Each node $v \in \mathcal{V}$ corresponds to a feature vector $\mathbf{x}_v \in \mathbb{R}^{d_{in}}$ where d_{in} is the number of input features. Let $\mathbf{X} = (\mathbf{x}_{v_1}, \cdots, \mathbf{x}_{v_n})^\top \in \mathbb{R}^{n \times d_{in}}$ be the node feature matrix composed of feature vectors. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be the adjacency matrix of $\mathcal{G}$. $\mathbf{A}_{i,j} = 1$ if $e_{i,j} \in \mathcal{E}$. Let $\mathbf{C} = (\mathbf{c}_{v_1}, \cdots, \mathbf{c}_{v_n})^\top \in \mathbb{R}^{n \times c}$ be the node label matrix composed of label vectors, where c is the number of labels, $\mathbf{C}_{i,j} \in \{0, 1\}$. Although we take node classification as an example in this paper, our method can be applied to general graph learning tasks (Appendix A).

Modules in GNN. Let $\texttt{GNN}(\mathcal{G}) = (\texttt{F}_\texttt{out} \circ \texttt{F}_\texttt{g} \circ \texttt{F}_\texttt{in})(\mathcal{G})$ be a GNN model. The target of $\texttt{GNN}(\mathcal{G})$ is to optimize the sets of learnable parameters $\mathcal{W}_\texttt{in}$, $\mathcal{W}_\texttt{g}$, and $\mathcal{W}_\texttt{out}$ and form the optimal mapping for $\texttt{F}_\texttt{in}$, $\texttt{F}_\texttt{g}$, and $\texttt{F}_\texttt{out}$. The input module $\texttt{F}_\texttt{in} : \mathbb{R}^{d_{in}} \mapsto \mathbb{R}^d$ maps input features to the hidden space, yielding $\mathbf{H}^{(0)} = \texttt{F}_\texttt{in}(\mathbf{X}; \mathcal{W}_\texttt{in})$, where d denotes the hidden dimensionality and ";" separates module input from the parameters. The module $\texttt{F}_\texttt{g} : \mathbb{R}^d \times \mathbb{R}^{n \times n} \mapsto \mathbb{R}^d$ applies general graph processing methods (Kipf & Welling, 2017; Rampášek et al., 2022), giving $\mathbf{H}^{(L)} = \texttt{F}_\texttt{g}(\mathbf{H}^{(0)}, \mathbf{A}; \mathcal{W}_\texttt{g})$ with L layers. Finally, the output module $\texttt{F}_\texttt{out} : \mathbb{R}^d \mapsto \mathbb{R}^c$ performs prediction, giving $\hat{\mathbf{C}} = \texttt{F}_\texttt{out}(\mathbf{H}^{(L)}; \mathcal{W}_\texttt{out})$.

Specifically, the focus of this paper is to model the mappings of the I/O modules $\texttt{F}_\texttt{in}$ and $\texttt{F}_\texttt{out}$, which can be uniformly formulated as $\texttt{F}_\texttt{i/o} : \mathbb{R}^{d_\texttt{src}} \mapsto \mathbb{R}^{d_\texttt{tgt}}$, yielding $\texttt{F}_\texttt{i/o}(\mathbf{H}; \mathcal{W}_\texttt{i/o})$. The input and output of the modules are termed as "source" and "target" to distinguish from the input and output of the whole model. $\texttt{F}_\texttt{i/o}$ as input modules has $d_\texttt{src} = d_\texttt{in}$, $d_\texttt{tgt} = d$, $\mathbf{H} = \mathbf{X}$, $\mathcal{W}_\texttt{i/o} = \mathcal{W}_\texttt{in}$. $\texttt{F}_\texttt{i/o}$ as output modules has $d_\texttt{src} = d$, $d_\texttt{tgt} = c$, $\mathbf{H} = \mathbf{H}^{(L)}$, $\mathcal{W}_\texttt{i/o} = \mathcal{W}_\texttt{out}$.

3.2 Mappings of the I/O Modules

To decouple the model parameters from specific datasets, we start by formulating the module mappings as dimension relations. Note that the nonlinearity in module mappings is obtained via

element-wise operations, which are independent of the I/O dimensionality. Therefore, we only focus on the linear part of the module mappings. Specifically, a linear mapping defines a projection from the source dimensions onto a new set of target dimensions (Fig. 2(a)). Learning the dimension relations enables the model to infer the optimal mapping from the source space to the target space.

Theorem 3.1 (Mapping with Dimension Relations)**.** *Given any linear mapping* $\mathbf{W} \in \mathbb{R}^{d_{\texttt{src}} \times d_{\texttt{tgt}}}$ *and* $s \in \mathbb{N}^+$*, there always exist two representation matrices* $\mathbf{S}_{\texttt{src}} \in \mathbb{R}^{d_{\texttt{src}} \times s}$ *and* $\mathbf{S}_{\texttt{tgt}} \in \mathbb{R}^{d_{\texttt{tgt}} \times s}$*, such that* $\mathbf{W} = \psi(\mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}})$*, where* $\psi(\cdot, \cdot)$ *is a bilinear composition function.*

This theorem shows that we can model the mappings as relations. A typical choice of $\psi(\cdot, \cdot)$ is the inner product form, where $\psi(\mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}}) = \mathbf{S}_{\texttt{src}} \mathbf{S}_{\texttt{tgt}}^\top$. Proof of Theorem 3.1 is presented in Appendix E.1. Without loss of generality, the I/O modules can be formulated as

$$\mathtt{F}_{\mathtt{i/o}}\left(\mathbf{H}; \mathcal{W}_{\mathtt{i/o}}\right) = \sigma\left[\mathbf{HW}\right] = \sigma\left[\mathbf{H}\psi(\mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}})\right] = \sigma\left[\mathbf{H}\mathbf{S}_{\texttt{src}}\mathbf{S}_{\texttt{tgt}}^\top\right], \tag{1}$$

where σ can be any nonlinear function.

Decomposition as Semantics. $\mathbf{S}_{\texttt{src}}$ and $\mathbf{S}_{\texttt{tgt}}$ are the decomposition results of the original weight matrix $\mathbf{W}$, which can be interpreted as the semantic embeddings associated with the dimensions of the source and target spaces, respectively. Each row of $\mathbf{S}_{\texttt{src}}$ and $\mathbf{S}_{\texttt{tgt}}$ encodes the specific semantic meaning of a dimension, typically characterizing graph nodes and their associated labels. For example, CoauthorCS (Shchur et al., 2019) provides node features representing the frequency of the paper keywords for each author's papers. These semantic descriptions can be embedded into a semantic space $\mathbb{R}^s$, giving rise to semantic embeddings such as $\mathbf{S}_{\texttt{src}}$ and $\mathbf{S}_{\texttt{tgt}}$.

Problems in Traditional Solutions. In traditional graph learning solutions (Zhou et al., 2020), $\mathtt{F}_{\mathtt{i/o}}$ is highly sensitive to the source and target spaces. It directly treats the space semantics as parameters for mapping, where $\mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}} \in \mathcal{W}_{\mathtt{i/o}}$. For instance, a single-layer perceptron can be formulated as $\sigma\left(\mathbf{H}\mathbf{S}_{\texttt{src}}\mathbf{S}_{\texttt{tgt}}^\top; \mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}}\right)$. Consequently, the learned parameter set $\mathcal{W}_{\mathtt{i/o}}$ becomes intrinsically tied to the specific source and target spaces, with its values tailored to particular spaces and its quantity scales to the number of dimensions in those spaces. This inherent sensitivity significantly limits the adaptability of pure GNNs to diverse datasets.

Our Solution. To address this issue, we propose to decouple the parameters $\mathcal{W}_{\mathtt{i/o}}$ from the source and target dimension semantics (Fig. 2), where feature semantics are redefined as parametric relations and label semantics are sampled uniformly in the semantic space.

3.3 Source-adaptive Input Module

A input module $\mathtt{F}_{\mathtt{in}}$ maps the input features to the hidden space. Based on Eq. 1, $\mathtt{F}_{\mathtt{in}}$ can be formulated as $\mathtt{F}_{\mathtt{in}}(\mathbf{X}) = \sigma[\mathbf{X}\mathbf{W}_{\mathtt{in}}] = \sigma[\mathbf{X}\mathbf{S}_{\texttt{src}}^{(\texttt{in})}\mathbf{S}_{\texttt{tgt}}^{(\texttt{in})\top}]$, where $\mathbf{S}_{\texttt{tgt}}^{(\texttt{in})} \in \mathbb{R}^{d \times s}$ is a learnable parameter matrix. Our focus is modeling the source space semantics $\mathbf{S}_{\texttt{src}}^{(\texttt{in})} \in \mathbb{R}^{d_{\texttt{in}} \times s}$ regarding specific inputs, *i.e.*, the semantics of the input features $\mathbf{X}$. To decouple the parameters from the number of source dimensions, $\mathbf{S}_{\texttt{src}}^{(\texttt{in})}$ can be formulated as a parametric function of $\mathbf{X}$:

$$\mathtt{f}_{\mathtt{in}} : \mathbb{R}^{n \times d_{\texttt{in}}} \mapsto \mathbb{R}^{d_{\texttt{in}} \times s}, \quad \mathbf{S}_{\texttt{src}}^{(\texttt{in})} = \mathtt{f}_{\mathtt{in}}(\mathbf{X}; \mathcal{W}_{\mathtt{in}}), \tag{2}$$

where $\mathtt{f}_{\mathtt{in}}(\cdot)$ is subject to two conditions: (1) Permutation invariance to the order of input nodes and equivariance to that of source dimensions; (2) Size independence of the parameter set $\mathcal{W}_{\mathtt{in}}$ to the values of n and $d_{\mathtt{in}}$. Input edges are disregarded in Eq. 2 as our focus lies in unifying the input module across different node features, while the unification of graph structures is left for future work.

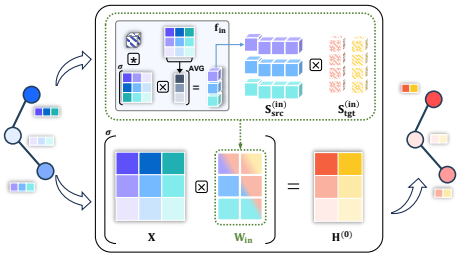


Figure 3: **Pipeline for the Unified Input Module.** A parametric relation function is employed to construct the source space semantics $\mathbf{S}_{\texttt{src}}^{(\texttt{in})}$, *i.e.*, the semantics of input features $\mathbf{X}$. The target space semantics $\mathbf{S}_{\texttt{tgt}}^{(\texttt{in})}$ is a learnable parameter matrix.

Modeling Features as Sets. Given the absence of graph structures and the permutation conditions for Eq. 2, the input features can be modeled as a set of channels $\{\mathbf{X}_{\cdot,j}\}$, where each channel corresponds to a set of nodes $\{\mathbf{X}_{i,j}\}$. As a result, Eq. 2 is transformed into a set-learning problem at both the channel level and the node level. Based on the universal functions for set learning (Zaheer et al.,

2017), $\mathtt{f}_{\mathtt{in}}(\cdot)$ can be formulated as follows. See Appendix E.2 for detailed derivation.

$$\mathtt{f}_{\mathtt{in}}(\mathbf{X}) = \sigma\left[\Theta\rho(\mathbf{X}^\top)\mathbf{1}\alpha^\top\right], \quad \Theta = \frac{d_{\mathtt{in}}\mathbf{X}^\top\mathbf{X}}{\mathbf{X}^\top\mathbf{X}\mathbf{1}}. \tag{3}$$

$\Theta \in \mathbb{R}^{d_{\mathtt{in}} \times d_{\mathtt{in}}}$ denotes the channel mixer that provides global information. $\rho(\cdot)$ computes the channel representations. $\mathbf{1}$ denotes the all-one vector and $\alpha \in \mathbb{R}^{s\times 1}$ is a learnable vector. The multiplication of α enables different activations through the nonlinearity of $\sigma(\cdot)$, giving s row embeddings in $\mathbf{S}^{(\mathtt{in})}_{\mathtt{src}}$. Although Zaheer et al. also provides an implementation named Deep Sets based on the universal set function, our input module differs from this specific implementation regarding problem formulation, conditions, and operator selections. Please refer to Appendix E.2 for a detailed discussion.

Unification via Relations. A direct implementation for $\rho(\cdot)$ is to take the specific values in $\mathbf{X}$ as channel representations. However, identical numerical values across different source spaces may correspond to entirely different semantics, making it nontrivial to uniformly map these values into a common semantic space via $\mathtt{f}_{\mathtt{in}}$. To tackle this, we propose modeling the channel relations as a proxy for feature semantics. By applying a shared relation function on feature channels, $\rho(\cdot)$ measures the same relation patterns across diverse source spaces. Crucially, the extracted patterns, such as similarity and co-variation, carry consistent semantic meaning. This consistency makes them naturally comparable and provides a stable foundation for unified semantic mapping. Among the typical choices for relation measure, such as Euclidean distance and inner product, we implement $\rho(\cdot)$ with the scaled product for its computing efficiency and training stability, giving $\rho(\mathbf{X}^\top) = \mathbf{X}^\top\mathbf{X}/\sqrt{n}$.

Source-adaptive Input. Compiling Eq. 1-3 gives rise to the input module as

$$\mathbf{H}^{(0)} = \mathtt{F}_{\mathtt{in}}\left(\mathbf{X}; \mathbf{S}^{(\mathtt{in})}_{\mathtt{tgt}}, \alpha\right) = \sigma\left[\mathbf{X}\mathbf{W}_{\mathtt{in}}\right] = \sigma\left[\mathbf{X}\mathtt{f}_{\mathtt{in}}(\mathbf{X})\mathbf{S}^{(\mathtt{in})\top}_{\mathtt{tgt}}\right], \tag{4}$$

where the mapping changes with specific input features, forming source-adaptive input for different datasets. $\mathtt{F}_{\mathtt{in}}$ is permutation invariant to the order of the source dimensions (see Appendix E.3) and has the parameter quantity independent of $d_{\mathtt{in}}$.

3.4 Target-insensitive Output Module

An output module $\mathtt{F}_{\mathtt{out}}$ maps the hidden representations to the label space. Based on Eq. 1, $\mathtt{F}_{\mathtt{out}}$ can be formulated as $\mathtt{F}_{\mathtt{out}}(\mathbf{H}^{(L)}) = \sigma[\mathbf{H}^{(L)}\mathbf{W}_{\mathtt{out}}] = \sigma[\mathbf{H}^{(L)}\mathbf{S}^{(\mathtt{out})}_{\mathtt{src}}\mathbf{S}^{(\mathtt{out})\top}_{\mathtt{tgt}}]$, where $\mathbf{S}^{(\mathtt{out})}_{\mathtt{src}} \in \mathbb{R}^{d\times s}$ is a learnable parameter matrix. Unifying the output module requires the modeling of the target space semantics $\mathbf{S}^{(\mathtt{out})}_{\mathtt{tgt}}$, *i.e.*, the semantics of the labels. However, the label knowledge of downstream datasets is typically unavailable. To tackle this problem, we propose a two-step approach, including prediction and assignment. In the prediction step, $\mathtt{F}_{\mathtt{out}}$ uniformly samples c points in $\mathbb{R}^s$ as pseudo-label semantics. By learning the relations between the parameter matrix $\mathbf{S}^{(\mathtt{out})}_{\mathtt{src}}$ and the pseudo-label semantics, the module can make predictions without prior label knowledge. In the assignment step, the module assigns the pseudo labels to the actual labels of each dataset, enabling precise adaptation to diverse target spaces.

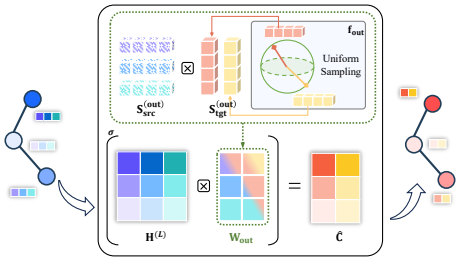


Figure 4: **Pipeline for the Unified Output Module.** Target space semantics $\mathbf{S}^{(\mathtt{out})}_{\mathtt{tgt}}$, *i.e.*, the semantics of pseudo labels, are selected uniformly in the semantic space. The source space semantics $\mathbf{S}^{(\mathtt{out})}_{\mathtt{src}}$ is a learnable parameter matrix.

Target-insensitive Prediction. The pseudo labels are independent of specific datasets, so their relations should remain consistent, without assuming that some labels are inherently closer than others. To ensure this, we uniformly sample pseudo labels in the semantic space. Note that the output module implements the relation function ψ as an inner product, quantifying relations by the angles between vectors. Accordingly, pseudo-label semantics are uniformly sampled on the unit sphere and mapped into Cartesian coordinates. In the s-dimensional spherical coordinates, a pseudo-label semantic vector can be denoted as $(1, \theta_1, \cdots, \theta_{s-1})$ with $\theta_i \in [0, \pi]$. To ensure the coverage of the unit sphere and maintain a uniform density of the semantics across all dimensions, we sample a number of $c^{1/(s-1)}$ values of equal intervals in $[0, \pi]$ for each θ_i and consider all combinations of these values. This results in a number of c semantic vectors distributed uniformly

on the s-dimensional unit sphere. The j-th semantic vector can be converted into the Cartesian coordinates with $\mathbf{S}^{(\texttt{out})}_{\texttt{tgt}_{j,i}} = \sigma[\prod_{k=1}^{i-1} \texttt{sin}^{\texttt{sgn}(i>1)}(\theta_k)\texttt{cos}^{\texttt{sgn}(i<s)}(\theta_i)],\ i \in \{1, \cdots, s\}$ (Blumenson, 1960). $\texttt{sgn}$ denotes the sign function, $\texttt{sgn}(\texttt{True}) = 1$ and $\texttt{sgn}(\texttt{False}) = 0$. Compiling the space division process as $\mathbf{S}^{(\texttt{out})}_{\texttt{tgt}} = \texttt{f}_{\texttt{out}}(c, s)$, the output module can be formulated as

$$\hat{\mathbf{C}} = \texttt{F}_{\texttt{out}}\left(\mathbf{H}^{(L)}; \mathbf{S}^{(\texttt{out})}_{\texttt{src}}\right) = \sigma\left[\mathbf{H}^{(L)}\mathbf{W}_{\texttt{out}}\right] = \sigma\left[\mathbf{H}^{(L)}\mathbf{S}^{(\texttt{out})}_{\texttt{src}}\texttt{f}_{\texttt{out}}(c, s)^{\top}\right]. \tag{5}$$

The implementation of the nonlinear function σ depends on the specific tasks (Appendix A).

Pseudo Label Assignment. The prediction step enables the output module to make predictions without prior knowledge of the target labels. When the label knowledge is available, one can further assign the pseudo labels to the real labels. In this paper, we consider two assignment strategies. (1) To explore the potential of pre-trained pure GNNs on general downstream tasks, observed target labels are only included during inference to match pseudo labels with real labels. Given the observed real labels $\mathbf{C}$, the mapping relations between the pseudo labels and the real labels can be formulated as $\hat{\mathbf{C}}\mathbf{P}(c\mathbf{I} - \mathbf{1}\mathbf{1}^{\top}) = c\texttt{log}(c\mathbf{C})$. The assignment matrix $\mathbf{P}$ can be solved as the least-squares solution of the linear equation without additional training. For detailed derivation, please refer to Appendix E.4. (2) To ensure fair comparison with LM-based graph models in the zero-shot setting, we replace observed labels with label knowledge by employing the embedded label semantics $\mathbf{S}^{\texttt{LM}}_{\texttt{tgt}}$ from language models (Li et al., 2024b) and construct the assignment matrix with $\texttt{f}_{\texttt{out}}(c, s)\mathbf{S}^{\texttt{LM}^{\top}}_{\texttt{tgt}}$.

3.5 Pipeline and Pre-training Strategy

GNNs with unified I/O take a mini-batch from a single dataset as input and are optimized on different datasets sequentially. During pre-training, GNNs only perform the first prediction step in the output module. Consequently, traditional optimization objectives that require strict alignment between the ordering of outputs and labels become inapplicable, such as cross-entropy loss and mean squared error. To address this problem, we draw inspiration from contrastive loss (Qiu et al., 2020) and propose to optimize the predicted distributions within the same class and across different classes. Given the node sets of each class $\{\mathcal{V}_1, \cdots, \mathcal{V}_c\}$, the loss function is formulated as

$$\mathcal{L} = \frac{1}{n}\left(\underbrace{\sum_{k,i,j,v_j \in \mathcal{V}_i}\left|\bar{\mathbf{C}}_{i,k} - \hat{\mathbf{C}}_{j,k}\right|}_{\mathcal{L}_{\texttt{inner}}} + \underbrace{2 - \frac{1}{c-1}\sum_{k,i,j,v_j \notin \mathcal{V}_i}\left|\bar{\mathbf{C}}_{i,k} - \hat{\mathbf{C}}_{j,k}\right|}_{\mathcal{L}_{\texttt{intra}}}\right), \bar{\mathbf{C}}_{i,\cdot} = \frac{1}{|\mathcal{V}_i|}\sum_{j,v_j \in \mathcal{V}_i}\hat{\mathbf{C}}_{j,\cdot}, \tag{6}$$

where $\bar{\mathbf{C}} \in \mathbb{R}^{c \times c}$ denotes the average prediction of each class. The inner-class loss, $\mathcal{L}_{\texttt{inn}}$, minimizes the prediction variance within the same class by ensuring that they are close to their class average. In contrast, the intra-class loss, $\mathcal{L}_{\texttt{int}}$, encourages differences in predictions across classes. To ensure positivity, $\mathcal{L}_{\texttt{int}}$ employs a constant bias of 2.

4 Experiment

4.1 Unified I/O Enables Pre-training with Pure GNNs

We now evaluate pre-trained GNNs with unified I/O modules, focusing on (1) whether pure GNNs can be pre-trained on diverse datasets and directly adapted to downstream datasets; (2) how pure GNNs generalize under different conditions, *i.e.*, the amount of training data and parameters, and the gap between the training and inference domains. Various real-world datasets of different scales are adopted from four domains (Tab. S11), including electronic commerce (e-com.), citation, social, and Wikipedia (wiki) graphs. Eight GNN methods are employed as the backbone, including GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018a), GIN (Xu et al., 2019), MixHop (Abu-El-Haija et al., 2019), DeepGCN (Li et al., 2019), GraphGPS (Rampášek et al., 2022), and $\mathbf{N^2}$ (Sun et al., 2024). Detailed experimental setup can be found in Appendix C.

Non-textual Datasets. To evaluate the pre-trained graph models on traditional datasets with non-textual attributes, we conduct a comparison with models that can make inference on downstream

Table 1: **Evaluation Results on Non-textual Datasets (Measured by accuracy except ROC AUC for tolokers: %).** Bold values denote the best results per test dataset. SUP denotes the best-performing results among supervised baselines. SSL denotes the best-performing results among self-supervised baselines. LP, Lap, and Rand denote Label Propagation, Laplacian decomposition, and random projection, respectively.

	Amazon Computers	Cora	Coauthor Physics	arxiv -year	twitch -gamer	tolokers	cha-meleon	actor	Avg. Rank
original split									
SUP	**91.09**$_{\pm 0.13}$	81.80$_{\pm 0.28}$	**95.52**$_{\pm 0.20}$	**48.03**$_{\pm 0.41}$	**61.09**$_{\pm 0.32}$	**80.72**$_{\pm 0.26}$	61.74$_{\pm 0.30}$	31.34$_{\pm 0.20}$	**1.87**
SSL	89.28$_{\pm 0.34}$	81.50$_{\pm 0.66}$	92.32$_{\pm 0.08}$	41.63$_{\pm 0.27}$	59.19$_{\pm 0.05}$	75.92$_{\pm 0.18}$	61.01$_{\pm 0.72}$	27.61$_{\pm 0.27}$	5.62
FUG	88.22$_{\pm 0.09}$	30.70$_{\pm 0.96}$	91.09$_{\pm 0.72}$	42.54$_{\pm 0.30}$	58.16$_{\pm 0.27}$	75.95$_{\pm 0.34}$	22.59$_{\pm 0.06}$	25.53$_{\pm 0.11}$	7.15
GraphAny	82.94$_{\pm 0.82}$	79.41$_{\pm 0.35}$	92.43$_{\pm 0.21}$	38.36$_{\pm 0.53}$	59.96$_{\pm 0.02}$	78.16$_{\pm 0.18}$	61.84$_{\pm 0.81}$	28.75$_{\pm 0.69}$	5.25
Label Propagation	87.27$_{\pm -}$	81.53$_{\pm -}$	95.67$_{\pm -}$	17.02$_{\pm -}$	58.30$_{\pm -}$	71.9$_{\pm -}$	18.86$_{\pm -}$	18.82$_{\pm -}$	6.87
SVD	83.89$_{\pm 0.76}$	79.92$_{\pm 0.56}$	92.87$_{\pm 0.55}$	42.54$_{\pm 0.22}$	59.73$_{\pm 0.06}$	76.12$_{\pm 0.26}$	62.04$_{\pm 0.85}$	30.05$_{\pm 0.13}$	4.45
Lap	84.98$_{\pm 0.78}$	78.33$_{\pm 0.84}$	91.11$_{\pm 0.78}$	42.18$_{\pm 0.14}$	59.05$_{\pm 0.12}$	76.40$_{\pm 0.20}$	60.66$_{\pm 0.55}$	24.91$_{\pm 0.02}$	6.62
Rand	88.85$_{\pm 1.65}$	81.64$_{\pm 1.48}$	91.75$_{\pm 2.79}$	41.37$_{\pm 1.26}$	59.03$_{\pm 1.85}$	76.02$_{\pm 1.34}$	61.22$_{\pm 1.86}$	34.39$_{\pm 1.08}$	5.12
Unified I/O	89.85$_{\pm 0.18}$	**82.32**$_{\pm 0.97}$	92.85$_{\pm 0.48}$	42.58$_{\pm 0.17}$	59.99$_{\pm 0.05}$	76.44$_{\pm 0.44}$	**62.13**$_{\pm 0.43}$	**35.35**$_{\pm 0.26}$	2.00
1-shot for Training-free Inference									
SUP	36.80$_{\pm 0.53}$	32.00$_{\pm 0.98}$	53.44$_{\pm 0.99}$	26.87$_{\pm 0.75}$	54.83$_{\pm 0.68}$	66.17$_{\pm 0.73}$	30.18$_{\pm 0.74}$	23.44$_{\pm 0.59}$	6.50
SSL	55.67$_{\pm 0.36}$	42.80$_{\pm 0.35}$	77.86$_{\pm 0.12}$	29.76$_{\pm 0.12}$	57.11$_{\pm 0.05}$	67.82$_{\pm 1.02}$	25.07$_{\pm 2.08}$	20.72$_{\pm 0.89}$	4.62
FUG	27.26$_{\pm 0.29}$	41.83$_{\pm 1.03}$	67.70$_{\pm 1.35}$	27.58$_{\pm 0.48}$	49.93$_{\pm 0.48}$	56.86$_{\pm 0.93}$	23.39$_{\pm 0.36}$	22.83$_{\pm 0.17}$	7.25
GraphAny	**62.87**$_{\pm 0.31}$	**53.63**$_{\pm 0.49}$	80.81$_{\pm 0.78}$	25.03$_{\pm 0.58}$	49.65$_{\pm 0.41}$	52.59$_{\pm 0.38}$	28.51$_{\pm 0.43}$	19.80$_{\pm 0.54}$	5.50
Label Propagation	46.26$_{\pm 2.95}$	18.81$_{\pm 1.55}$	22.98$_{\pm 2.88}$	18.54$_{\pm 0.01}$	53.00$_{\pm 1.00}$	66.20$_{\pm 2.92}$	18.86$_{\pm 0.62}$	11.45$_{\pm 0.69}$	8.12
SVD	55.42$_{\pm 0.31}$	42.22$_{\pm 0.83}$	76.31$_{\pm 0.12}$	33.44$_{\pm 0.25}$	56.94$_{\pm 0.10}$	67.62$_{\pm 0.49}$	31.14$_{\pm 0.18}$	23.61$_{\pm 0.06}$	4.15
Lap	55.53$_{\pm 1.21}$	43.73$_{\pm 0.43}$	78.41$_{\pm 1.26}$	33.08$_{\pm 0.35}$	57.00$_{\pm 0.09}$	67.62$_{\pm 0.58}$	29.82$_{\pm 0.16}$	21.89$_{\pm 0.03}$	4.15
Rand	58.80$_{\pm 1.85}$	42.34$_{\pm 1.42}$	76.05$_{\pm 2.27}$	32.73$_{\pm 1.03}$	57.72$_{\pm 1.75}$	67.98$_{\pm 2.07}$	30.75$_{\pm 0.91}$	24.71$_{\pm 0.74}$	3.37
Unified I/O	59.89$_{\pm 0.80}$	43.94$_{\pm 0.50}$	**85.13**$_{\pm 0.68}$	**33.47**$_{\pm 0.14}$	**57.90**$_{\pm 0.15}$	**68.51**$_{\pm 0.13}$	**32.00**$_{\pm 0.11}$	**25.69**$_{\pm 0.12}$	**1.25**
3-shot for Training-free Inference									
SUP	65.64$_{\pm 0.30}$	37.10$_{\pm 0.55}$	76.44$_{\pm 0.48}$	27.80$_{\pm 0.51}$	54.02$_{\pm 0.62}$	68.89$_{\pm 0.81}$	33.57$_{\pm 0.41}$	20.88$_{\pm 0.43}$	5.62
SSL	64.51$_{\pm 2.68}$	48.73$_{\pm 1.86}$	84.22$_{\pm 0.12}$	26.53$_{\pm 1.11}$	56.78$_{\pm 0.16}$	59.24$_{\pm 2.24}$	31.99$_{\pm 0.37}$	20.42$_{\pm 0.15}$	5.25
FUG	50.59$_{\pm 1.43}$	47.77$_{\pm 1.21}$	66.52$_{\pm 2.47}$	24.02$_{\pm 0.34}$	49.83$_{\pm 0.18}$	57.93$_{\pm 1.17}$	25.47$_{\pm 0.56}$	20.35$_{\pm 0.33}$	7.75
GraphAny	**70.04**$_{\pm 0.70}$	**66.32**$_{\pm 0.61}$	**91.33**$_{\pm 0.79}$	24.74$_{\pm 0.80}$	54.71$_{\pm 0.81}$	54.12$_{\pm 0.63}$	33.69$_{\pm 0.73}$	18.55$_{\pm 0.72}$	4.37
Label Propagation	60.04$_{\pm 2.64}$	31.64$_{\pm 2.22}$	32.82$_{\pm 2.47}$	18.49$_{\pm 0.06}$	52.95$_{\pm 2.12}$	66.36$_{\pm 3.47}$	16.45$_{\pm 2.18}$	11.91$_{\pm 0.73}$	8.12
SVD	59.86$_{\pm 0.57}$	45.23$_{\pm 0.90}$	76.88$_{\pm 0.35}$	33.44$_{\pm 0.30}$	56.93$_{\pm 0.13}$	68.47$_{\pm 0.34}$	33.64$_{\pm 0.18}$	23.01$_{\pm 0.04}$	4.81
Lap	56.58$_{\pm 0.31}$	47.31$_{\pm 0.45}$	76.88$_{\pm 0.62}$	34.64$_{\pm 0.38}$	56.75$_{\pm 0.11}$	69.34$_{\pm 0.35}$	32.21$_{\pm 0.16}$	21.54$_{\pm 0.02}$	4.93
Rand	66.11$_{\pm 1.00}$	48.98$_{\pm 1.57}$	79.31$_{\pm 1.34}$	35.05$_{\pm 1.07}$	56.96$_{\pm 1.78}$	69.09$_{\pm 1.09}$	33.68$_{\pm 1.00}$	24.99$_{\pm 0.74}$	2.75
Unified I/O	68.33$_{\pm 0.28}$	49.21$_{\pm 0.91}$	84.68$_{\pm 0.67}$	**35.32**$_{\pm 0.29}$	**57.64**$_{\pm 0.06}$	**71.77**$_{\pm 0.25}$	**33.76**$_{\pm 0.11}$	**25.20**$_{\pm 0.11}$	**1.37**

datasets without additional training efforts, including Label Propagation (Kothari & Jain, 2002), GraphAny (Zhao et al., 2024a), and parameter-free feature alignment methods SVD-based (Sun et al., 2023), Laplacian-based (Sun et al., 2025), and random-based (Tang et al., 2024) input combined with our output module. We also include parameterized feature alignment method FUG (Zhao et al., 2024b), supervised baselines (GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018a)) and self-supervised baselines, including contrastive-based methods (DGI (Veličković et al., 2018b), GraphCL (You et al., 2020), GraphACL (Xiao et al., 2023), GRACE (Zhu et al., 2020), SimGRACE (Xia et al., 2022)) and reconstruction-based methods (MaskGAE (Li et al., 2023) and GraphMAE2 (Hou et al., 2023)). For FUG and self-supervised baselines, we follow GraphAny to solve the mapping matrix from the learned hidden representations to the labels in a training-free manner. The models are pre-trained on datasets from four distinct domains: amazon-ratings, ogbn-arxiv, Facebook, and roman-empire. The best results among the eight GNN backbones are reported for both our unified I/O and parameter-free feature alignment methods.

Results are summarized in Tab. 1. Compared to supervised baselines, pre-training with unified I/O achieves comparable or even superior performance on the original split. Under 1-shot and 3-shot settings, unified I/O consistently outperforms the supervised baselines. These results demonstrate that model pre-training is necessary for graph learning, particularly in data-scarce settings. Compared to other baselines, pre-trained GNNs with unified I/O obtain clear advantages on heterophilic datasets, whereas GraphAny performs better on homophilic datasets in certain cases. This difference reflects the higher transferability of node-feature knowledge, compared to structural knowledge learned by GNNs with unified I/O (see Appendix D.4 for details). Nevertheless, pure GNNs with unified I/O achieve the best average rank across different datasets. This indicates the superior ability of our unified I/O to support effective pre-training and downstream adaptation.

Textual Datasets. In comparison to the LM-based models, we adopt textual datasets (Chen et al., 2024c) for pre-training and inference. Models are pre-trained on PubMed, bookhistory, amazon-ratings, and arxiv with textual features. To ensure fair comparison, pure GNNs employ the second pseudo label assignment strategy in Sec 3.4. Tab. 2 presents comparison results with methods employing LM as I/O (OFA (Liu et al., 2023a), ZeroG (Li et al., 2024b), LLaGA (Chen et al., 2024a), GraphCLIP (Zhu et al., 2025), and RiemannGFM (Sun et al., 2025)), and parameter-free feature

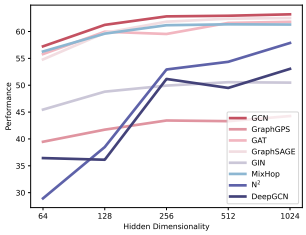
Table 2: **Evaluation Results on Textual Datasets (Measured by accuracy: %).** Bold values denote the best results per test dataset. LP, Lap, and Rand denote Label Propagation, Laplacian decomposition, and random projection, respectively.

	WIKICS	BOOKCHILD	COMPUTERS	PHOTO	SPORTSFIT	PRODUCTS	DBLP	TOLOKERS
LLAGA	2.65±0.07	21.05±0.16	23.00±0.04	5.10±0.04	5.45±0.24	10.40±0.02	11.55±0.07	71.80±0.28
ZEROG	**37.13**±0.41	12.62±2.25	6.72±0.52	3.84±0.17	30.04±0.20	**24.79**±1.68	**54.86**±0.52	78.13±0.01
GRAPHCLIP	3.54±0.51	16.16±0.01	25.43±0.16	4.19±0.01	7.61±0.10	9.31±0.08	34.94±0.37	78.01±0.07
OFA	33.89±0.09	1.98±0.01	5.98±0.10	14.72±0.01	12.65±0.01	2.64±0.02	51.01±0.02	78.16±0.01
RIEMANNGFM	4.26±0.11	1.74±0.06	5.77±0.01	6.93±0.49	3.83±0.02	2.45±0.05	38.68±0.37	77.65±0.01
SVD	30.24±0.37	16.16±0.34	27.65±0.43	49.15±0.98	41.93±1.04	13.04±0.20	46.91±0.60	78.50±0.52
LAP	23.43±0.24	30.47±0.82	27.05±0.38	49.09±0.64	42.05±1.43	13.05±0.28	43.63±0.39	78.47±0.58
RAND	23.41±0.36	30.32±1.01	29.82±0.38	49.04±1.22	41.83±1.13	13.42±0.78	47.06±0.77	78.16±0.83
UNIFIED I/O	32.63±0.21	**31.60**±0.69	**30.82**±0.26	**49.20**±0.31	**42.87**±1.03	15.93±0.32	52.04±0.86	**78.66**±0.49

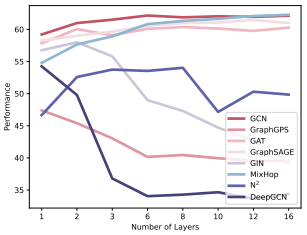
alignment methods. Pure GNNs with our unified I/O achieve superior performance to baselines except for ZeroG on WikiCS, products, and DBLP. Notably, baseline models with LM introduce knowledge gained from enormous training data for graph learning. In contrast, pure GNNs with only the knowledge of PubMed, bookhistory, amazon-ratings, and arxiv during pre-training achieve better performance. Compared to ZeroG, which is restricted to graphs with rich textual attributes, pure GNNs can be applied to either textual or non-textual datasets. This demonstrates the potential of pre-training with pure GNNs in tackling various graph tasks.

4.2 PRE-TRAINING CONDITION STUDY

Pre-training with Scaling Parameters. Pure GNNs are pre-trained on amazon-ratings, ogbn-arxiv, Facebook, and roman-empire, with scaling parameters. To mitigate the impact of the domain gap, one dataset is selected from each domain, encompassing both homophilic and heterophilic graphs. Performance results are averaged across test graphs (Tab. S11). Fig. 5(a) presents the impact of hidden dimensionality d, showing that larger values of d consistently improve performance across all GNN architectures. In contrast, the impact of the number of layers L varies by architecture. As shown in Fig.5(b), deeper network configurations consistently enhance performance for GCN, GAT, GraphSAGE, and MixHop, whereas GIN, DeepGCN, and GraphGPS experience performance degradation with more layers. For $\mathbf{N^2}$, performance initially improves with more layers but eventually declines when L exceeds 8. In Appendix D.2, we examine larger parameter configurations for GCN and GIN. The results show that increasing the number of layers leads to over-parameterization, whereas enlarging the hidden dimension does not. All these results suggest that scaling up hidden dimensionality is the prior strategy for enhancing pre-trained GNN models, while the configuration of the number of layers depends on the specific GNN architectures.



(a) Hidden Dimensions



(b) Layers

Figure 5: **Pre-training with Scaling Parameters.**

Pre-training with Scaling Data. To assess the effect of data scaling in GNN pre-training, we consider two strategies: (1) increasing the number of training datasets (one, three, and four combined), and (2) enlarging individual datasets (1k, 10k, and 100k nodes per dataset). Detailed combinations of the training and test datasets are provided in Appendix Tab. S11, S12. Due to the varying difficulty of specific downstream tasks, model performance cannot be directly compared across different test datasets. To address this problem, training datasets are split into three groups with an average number of nodes around 1k, 10k, and 100k. Model performance is then normalized with the group average performance. For more analysis on the data scaling strategies, please refer to Appendix D.3. The averaged results over 1,848 data points across different backbones and test datasets are presented in Fig. 6. We can see that increasing either the minimum dataset sizes or increasing training datasets with the same node scale improves the model performance. This suggests that scaling up training data enhances the adaptation ability of the pre-trained models to test datasets.

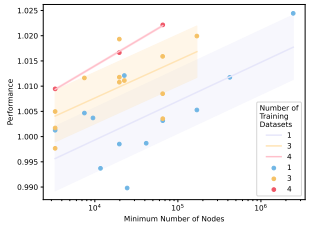
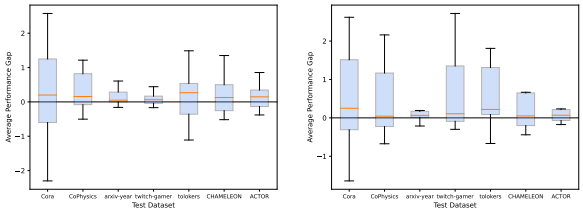


Figure 6: **Pre-training with Scaling Data.** Different dot colors indicate the number of datasets for pre-training. The lines denote trending-fitting with dots under the same number of dataset combinations.



(a) Different Domain (b) Different Heterophily

Figure 7: **Adaptation Results with Different Domain Gap.** The cross-domain performance gap is calculated as the performance difference between GNNs pre-trained on the same-domain group as the test datasets and those on the different-domain group. The heterophily performance gap is computed by subtracting the same homophily–heterophily training type as the test datasets with that of the opposite type.

Inference with Different Domain Gap. We further evaluate pre-trained GNNs on adaptation tasks: (1) pre-training in one domain and adapting to the others, (2) pre-training across three domains and adapting to the remaining one, and (3) transferring across datasets with varying homophily–heterophily. Dataset combinations are listed in Tab. S11, S12.

The adaptation results for tasks (1) and (2) are presented in Fig. 7(a). For each test dataset, the training datasets are categorized into two groups: those collected from the same domain as the test dataset and those from different domains. The performance gap is computed by subtracting the performance of GNNs pre-trained on the different-domain group from that of the same-domain group ($\texttt{Metric}_{\texttt{same}} - \texttt{Metric}_{\texttt{diff}}$). To avoid the influence of the dataset scales, the model performance is first grouped based on the average number of nodes per training dataset (1k, 10k, and 100k), and then averaged and compared within each group. As shown in Fig. 7(a), pre-trained GNNs generally exhibit positive performance gaps between the same-domain and different-domain groups across diverse test datasets. This observation indicates that aligning the training and test domains tends to improve model performance during inference.

The adaptation results for task (3) with different homophily-heterophily are presented in Fig. 7(b). When comparing on the same test dataset, the training datasets are categorized into two groups: those with the same homophily-heterophily as the test dataset and those with the opposite. The performance gap is the performance difference between GNNs pre-trained on the same-homophily-heterophily group and that of the opposite-homophily-heterophily group ($\texttt{Metric}_{\texttt{same}} - \texttt{Metric}_{\texttt{oppo}}$). Since training datasets of certain scales are missing for heterophily/homophily graphs in certain domains, our experimental analysis for task (3) directly averages results by mixing all training scales together. Fig. 7(b) shows that models pre-trained on datasets with the same homophily-heterophily as the test datasets tend to achieve better performance during inference. This suggests that one may construct a training dataset based on the homophily-heterophily of downstream tasks to gain better results. We also compared adaptation results with the same pre-training datasets in Appendix D.4. Results in Fig. 11(a) show that models pre-trained on either homophilic or heterophilic graphs gain better results on heterophilic graphs than homophilic graphs. This can be attributed to the better transferability of the node feature knowledge learned by our unified I/O than the structural knowledge (Fig. 11(d)), where node feature knowledge better benefits the adaptation to heterophilic graphs (Fig. 11(b)) and structural knowledge benefits homophilic tasks (Fig. 11(c)). Please refer to Appendix D.4 for more details.

4.3 Pre-trained Pure GNNs provide Competitive Graph Operators

Our unified I/O modules enable seamless adaptation of pure GNN architectures across diverse datasets. To further evaluate the effectiveness of the pre-trained GNN operators, we fine-tune the models pre-trained on amazon-ratings, ogbn-arxiv, Facebook, and roman-empire. The internal GNN module $\texttt{F}_{\texttt{g}}$ within the pre-trained models is frozen during the fine-tuning. The unification-oriented I/O function

Table 3: **Evaluation Results of Fine-tuning the I/O Modules in the Pre-trained GNNs (Measured by accuracy except ROC AUC for tolokers: %).** Bold values denote the best results per test dataset.

	CORA	PUBMED	AMAZON COMPUTERS	WIKICS	AMAZON -RATINGS	MINESWEEPER	TOLOKERS
SELF-SUPERVISED LEARNING AND SUPERVISED OUTPUT LEARNING							
DGI	84.00$_{\pm 0.28}$	83.73$_{\pm 0.41}$	82.11$_{\pm 0.16}$	75.05$_{\pm 0.39}$	40.80$_{\pm 0.74}$	88.45$_{\pm 0.38}$	77.73$_{\pm 0.14}$
GRACE	84.30$_{\pm 0.57}$	85.81$_{\pm 0.18}$	89.67$_{\pm 0.36}$	75.80$_{\pm 0.56}$	42.19$_{\pm 0.12}$	86.15$_{\pm 0.45}$	75.06$_{\pm 0.14}$
GRAPHACL	75.00$_{\pm 0.75}$	82.93$_{\pm 0.17}$	80.58$_{\pm 0.15}$	68.00$_{\pm 0.75}$	40.65$_{\pm 0.14}$	87.23$_{\pm 0.11}$	77.68$_{\pm 0.25}$
GRAPHCL	63.33$_{\pm 1.76}$	63.53$_{\pm 1.08}$	84.57$_{\pm 0.34}$	76.32$_{\pm 0.18}$	42.35$_{\pm 0.37}$	79.85$_{\pm 0.12}$	80.03$_{\pm 0.12}$
MASKGAE	75.13$_{\pm 1.78}$	75.27$_{\pm 1.03}$	92.15$_{\pm 0.05}$	78.25$_{\pm 0.20}$	43.54$_{\pm 0.30}$	84.33$_{\pm 0.16}$	81.13$_{\pm 0.34}$
SIMGRACE	67.07$_{\pm 0.82}$	77.63$_{\pm 0.82}$	87.44$_{\pm 0.18}$	78.75$_{\pm 0.28}$	43.46$_{\pm 0.23}$	84.23$_{\pm 0.16}$	80.29$_{\pm 0.30}$
GRAPHMAE2	79.50$_{\pm 0.51}$	67.07$_{\pm 0.91}$	91.03$_{\pm 0.19}$	76.24$_{\pm 0.11}$	40.95$_{\pm 0.71}$	80.16$_{\pm 0.10}$	80.17$_{\pm 0.07}$
SUPERVISED LEARNING							
GRAPHSAGE	78.83$_{\pm 0.50}$	88.11$_{\pm 0.05}$	91.09$_{\pm 0.02}$	78.13$_{\pm 0.15}$	45.71$_{\pm 0.38}$	90.55$_{\pm 0.10}$	83.06$_{\pm 0.59}$
GAT	77.51$_{\pm 2.35}$	85.30$_{\pm 0.15}$	89.78$_{\pm 0.02}$	76.35$_{\pm 0.80}$	44.54$_{\pm 0.52}$	82.07$_{\pm 1.17}$	77.37$_{\pm 0.28}$
GIN	77.36$_{\pm 0.15}$	85.13$_{\pm 0.55}$	90.51$_{\pm 0.80}$	74.02$_{\pm 0.62}$	46.33$_{\pm 0.11}$	74.93$_{\pm 0.58}$	60.93$_{\pm 2.25}$
GCN	80.35$_{\pm 0.25}$	85.44$_{\pm 0.50}$	90.66$_{\pm 0.13}$	78.55$_{\pm 0.01}$	46.71$_{\pm 0.25}$	76.43$_{\pm 1.05}$	77.79$_{\pm 0.12}$
GRAPHGPS	58.61$_{\pm 0.05}$	85.21$_{\pm 0.30}$	88.87$_{\pm 0.20}$	75.18$_{\pm 0.04}$	47.85$_{\pm 0.29}$	89.64$_{\pm 0.24}$	79.82$_{\pm 0.06}$
PRE-TRAINING AND I/O FINE-TUNING							
UNIFIED I/O	**84.63**$_{\pm 0.12}$	**88.91**$_{\pm 0.45}$	**92.33**$_{\pm 0.25}$	**78.98**$_{\pm 0.12}$	**51.59**$_{\pm 0.05}$	**91.39**$_{\pm 0.15}$	**83.29**$_{\pm 0.13}$

$\mathtt{f}_{\mathtt{in}}(\cdot)$ in Eq. 4 and $\mathtt{f}_{\mathtt{out}}(\cdot)$ in Eq. 5 are replaced with learnable parameters. The best fine-tuning results among different backbones on downstream graphs are summarized in Tab. 3. For full results, please refer to Appendix D.5. The pre-trained operators achieve superior performance compared to supervised methods and self-supervised methods (DGI (Veličković et al., 2018b), GRACE (Zhu et al., 2020), GraphACL (Xiao et al., 2023), GraphCL (You et al., 2020), SimGRACE (Xia et al., 2022), MaskGAE (Li et al., 2023), GraphMAE2 (Hou et al., 2023)). Notably, the pre-trained operators require minimal hyperparameter tuning, with only dropout adjusted during fine-tuning. This significantly simplifies the hyperparameter tuning process, enabling efficient adaptation of pre-trained GNNs to various graphs with promising performance.

4.4 COST ANALYSIS ON THE UNIFIED I/O

Both the space complexity and time complexity of our unified I/O are $O(n)$, with $d_{\mathtt{in}}, d, s, c \ll n$. Empirical time consumption and information loss results are provided in Appendix D.6 and D.7. Results show that unified I/O maintains a reasonable time cost under various scales of graphs and numbers of input features, and does not cause severe information loss. This demonstrates the effectiveness of our unified I/O in learning input and output mappings.

5 CONCLUSION

In this paper, we achieved unified input and output for graphs, enabling pre-training with pure GNNs across diverse datasets. To decouple learnable parameters from the number and semantics of input features and output labels, our unified I/O modules employ a shared relation function for the feature semantics and uniformly sampled points for the label semantics. By integrating our unified I/O modules with various GNN architectures, we demonstrated that pure GNNs can serve as effective graph learners for direct adaptation to downstream tasks and provide competitive pre-trained graph operators. For the usage of LLM and the limitation discussion, please refer to Appendix F and G.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of our work. Specifically, we provide a detailed description of the experimental setups in Appendix C, including evaluation settings, dataset information, architecture configurations, and hyperparameter setups. All datasets employed are publicly available. In addition, the code implementation of our proposed methods is provided in the supplementary material. The complete source code will be released publicly upon acceptance of the paper.

REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *International Conference on Machine Learning*, pp. 21–29, Long Beach, USA, 2019. PMLR.

L. E. Blumenson. A Derivation of n-Dimensional Spherical Coordinates. *The American Mathematical Monthly*, 67(1):63–66, 1960.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 1306–1313, Atlanta, Georgia, July 2010. AAAI Press.

Mark Chen, Jerry Tworek, and Heewoo et al. Jun. Evaluating Large Language Models Trained on Code, 2021.

Runjin Chen, Tong Zhao, Ajay Kumar Jaiswal, Neil Shah, and Zhangyang Wang. LLaGA: Large Language and Graph Assistant. In *International Conference on Machine Learning*, pp. 7809–7823. PMLR, 2024a.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, and Jiliang Tang. Exploring the Potential of Large Language Models (LLMs)in Learning on Graphs. *SIGKDD Explor. Newsl.*, 25(2):42–61, 2024b.

Zhikai Chen, Haitao Mao, Jingzhe Liu, Yu Song, Bingheng Li, Wei Jin, Bahare Fatemi, Anton Tsitsulin, Bryan Perozzi, Hui Liu, and Jiliang Tang. Text-space Graph Foundation Models: Comprehensive Benchmarks and New Insights. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7464–7492, December 2024c.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3213–3223, 2016.

Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. Universal Prompt Tuning for Graph Neural Networks. *Advances in Neural Information Processing Systems*, 36:52464–52489, 2023.

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

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *International Conference on Neural Information Processing Systems*, pp. 1025–1035, Red Hook, USA, 2017. Curran Associates Inc.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*, 2020.

Zhenyu Hou, Yufei He, Yukuo Cen, Xiao Liu, Yuxiao Dong, Evgeny Kharlamov, and Jie Tang. GraphMAE2: A Decoding-Enhanced Masked Self-Supervised Graph Learner. In *Proceedings of the ACM Web Conference 2023*, pp. 737–746, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3543507.3583379.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133. Curran Associates, Inc., 2020.

Qian Huang, Hongyu Ren, Peng Chen, Gregor Kržmanc, Daniel Zeng, Percy Liang, and Jure Leskovec. PRODIGY: Enabling In-context Learning Over Graphs. In *Conference on Neural Information Processing Systems*, 2023.

Renhong Huang, Jiarong Xu, Xin Jiang, Chenglu Pan, Zhiming Yang, Chunping Wang, and Yang Yang. Measuring Task Similarity and Its Implication in Fine-Tuning Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38 of *11*, pp. 12617–12625, 2024.

Yongcheng Jing, Chongbin Yuan, Li Ju, Yiding Yang, Xinchao Wang, and Dacheng Tao. Deep Graph Reprogramming. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24345–24354, 2023.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 5156–5165. PMLR, November 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference for Learning Representations*, San Diego, USA, 2015.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, Toulon, France, 2017.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment Anything, 2023.

Lecheng Kong, Jiarui Feng, Hao Liu, Chengsong Huang, Jiaxin Huang, Yixin Chen, and Muhan Zhang. GOFA: A Generative One-For-All Model for Joint Graph Language Modeling. In *The Thirteenth International Conference on Learning Representations*, October 2024.

R. Kothari and V. Jain. Learning from labeled and unlabeled data. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, volume 3, pp. 2803–2808 vol.3, 2002.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs Go As Deep As CNNs? In *IEEE/CVF International Conference on Computer Vision*, pp. 9266–9275, Seoul, South Korea, 2019. IEEE.

Jintang Li, Ruofan Wu, Wangbin Sun, Liang Chen, Sheng Tian, Liang Zhu, Changhua Meng, Zibin Zheng, and Weiqiang Wang. What's Behind the Mask: Understanding Masked Graph Modeling for Graph Autoencoders. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1268–1279, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3580305.3599546.

Shengrui Li, Xueting Han, and Jing Bai. AdapterGNN: Parameter-Efficient Fine-Tuning Improves Generalization in GNNs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13600–13608, 2024a.

Yuhan Li, Peisong Wang, Zhixun Li, Jeffrey Xu Yu, and Jia Li. ZeroG: Investigating Cross-dataset Zero-shot Transferability in Graphs. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1725–1735, New York, NY, USA, 2024b. Association for Computing Machinery.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In *Advances in Neural Information Processing Systems*, volume 34, pp. 20887–20902. Curran Associates, Inc., 2021.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One For All: Towards Training One Graph Model For All Classification Tasks. In *International Conference on Learning Representations*, 2023a.

Zemin Liu, Xingtong Yu, Yuan Fang, and Xinming Zhang. GraphPrompt: Unifying Pre-Training and Downstream Tasks for Graph Neural Networks. In *Proceedings of the ACM Web Conference 2023*, pp. 417–428, New York, NY, USA, 2023b. Association for Computing Machinery.

Péter Mernyei and Cătălina Cangea. Wiki-CS: A Wikipedia-Based Benchmark for Graph Neural Networks. In *International Conference on Machine Learning Workshop on Graph Representation Learning and Beyond*. arXiv, January 2022.

Carl D Meyer. *Matrix analysis and applied linear algebra*. SIAM, 2023.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *International Conference on Machine Learning Workshop on Graph Representation Learning and Beyond*, Virtual Only, 2020. arXiv.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and Red et al. Avila. GPT-4 Technical Report, 2024.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*, 2019.

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*, Kigali, Rwanda, 2023.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1150–1160, New York, NY, USA, 2020. Association for Computing Machinery.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. In *Advances in Neural Information Processing Systems*, volume 35, pp. 14501–14515, New Orleans, USA, 2022.

Benedek Rozemberczki, Carl Allen, Rik Sarkar, and xx Thilo Gross. Multi-Scale attributed node embedding. *Journal of Complex Networks*, 9(1):1–22, April 2021. ISSN 2051-1329.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29(3):93, 2008. ISSN 0738-4602, 0738-4602.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of Graph Neural Network Evaluation. *arXiv:1811.05868 [cs, stat]*, 2019.

Junshu Sun, Chenxue Yang, Xiangyang Ji, Qingming Huang, and Shuhui Wang. Towards Dynamic Message Passing on Graphs. In *Conference on Neural Information Processing Systems*, December 2024.

Li Sun, Zhenhao Huang, Suyang Zhou, Qiqi Wan, Hao Peng, and Philip S. Yu. RiemannGFM: Learning a Graph Foundation Model from Structural Geometry. In *THE WEB CONFERENCE 2025*, January 2025.

Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. GPPT: Graph Pre-training and Prompt Tuning to Generalize Graph Neural Networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1717–1727, New York, NY, USA, 2022. Association for Computing Machinery.

Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. All in One: Multi-Task Prompting for Graph Neural Networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2120–2131, New York, NY, USA, 2023. Association for Computing Machinery.

Zhen Tan, Ruocheng Guo, Kaize Ding, and Huan Liu. Virtual Node Tuning for Few-shot Node Classification. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2177–2188, New York, NY, USA, 2023. Association for Computing Machinery.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. GraphGPT: Graph Instruction Tuning for Large Language Models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 491–500, New York, NY, USA, 2024. Association for Computing Machinery.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, Vancouver, Canada, 2018a.

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2018b.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? In *International Conference on Neural Information Processing Systems*, pp. 30840–30861, Red Hook, NY, USA, 2023. Curran Associates Inc.

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998. ISSN 1476-4687. doi: 10.1038/30918.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*, pp. 6861–6871, Long Beach, USA, 2019. PMLR.

Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, and Stan Z. Li. SimGRACE: A Simple Framework for Graph Contrastive Learning without Data Augmentation. In *Proceedings of the ACM Web Conference 2022*, pp. 1070–1079, New York, NY, USA, 2022. Association for Computing Machinery. doi: 10.1145/3485447.3512156.

Teng Xiao, Huaisheng Zhu, Zhengyu Chen, and Suhang Wang. Simple and Asymmetric Graph Contrastive Learning without Augmentations. In *Advances in Neural Information Processing Systems*, volume 36, pp. 16129–16152, 2023.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, New Orleans, LA, USA, 2019.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 5812–5823, Red Hook, NY, USA, 2020. Curran Associates Inc.

Xingtong Yu, Yuan Fang, Zemin Liu, Yuxia Wu, Zhihao Wen, Jianyuan Bo, Xinming Zhang, and Steven C. H. Hoi. A Survey of Few-Shot Learning on Graphs: From Meta-Learning to Pre-Training and Prompt Learning, 2024.

Xingtong Yu, Zechuan Gong, Chang Zhou, Yuan Fang, and Hui Zhang. SAMGPT: Text-free Graph Foundation Model for Multi-domain Pre-training and Cross-domain Adaptation. In *THE WEB CONFERENCE 2025*, 2025.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Libo Zhang, Lutao Jiang, Ruyi Ji, and Heng Fan. PIDray: A Large-Scale X-ray Benchmark for Real-World Prohibited Item Detection. *International Journal of Computer Vision*, 131(12):3170–3192, 2023a.

Wen Zhang, Yushan Zhu, Mingyang Chen, Yuxia Geng, Yufeng Huang, Yajing Xu, Wenting Song, and Huajun Chen. Structure Pretraining and Prompt Tuning for Knowledge Graph Transfer. In *Proceedings of the ACM Web Conference*, pp. 2581–2590, New York, NY, USA, 2023b. Association for Computing Machinery.

Jianan Zhao, Zhaocheng Zhu, Mikhail Galkin, Hesham Mostafa, Michael M. Bronstein, and Jian Tang. Fully-inductive Node Classification on Arbitrary Graphs. In *The Thirteenth International Conference on Learning Representations*, 2024a.

Jitao Zhao, Di Jin, Meng Ge, Lianze Shan, Xin Wang, Dongxiao He, and Zhiyong Feng. FUG: Feature-Universal Graph Contrastive Pre-training for Graphs with Diverse Node Features. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep Graph Contrastive Representation Learning. In *International Conference on Machine Learning Workshop on Graph Representation Learning and Beyond*, 2020.

Yun Zhu, Haizhou Shi, Xiaotang Wang, Yongchao Liu, Yaoke Wang, Boci Peng, Chuntao Hong, and Siliang Tang. GraphCLIP: Enhancing Transferability in Graph Foundation Models for Text-Attributed Graphs. In *THE WEB CONFERENCE 2025*, 2025.

A General Target-insensitive Output Module

Sec. 3.4 takes node classification as an example to formulate our unified output module. In practice, our method can be applied to general classification and regression tasks. Specifically, the general output module in Eq. 5 can be formulated as

$$\hat{\mathbf{C}} = \mathtt{F}_{\mathtt{out}}\left(\mathbf{H}^{(L)}\right) = \sigma\left(\mathbf{g}(\mathbf{H}^{(L)})\mathbf{S}_{\mathtt{src}}^{(\mathtt{out})}\mathtt{f}_{\mathtt{out}}(c,s)^{\top}\right), \tag{S7}$$

where $\mathbf{g}$ denotes the read-out function for different levels of tasks. For classification tasks, the nonlinear function σ can be implemented as `softmax`, `sigmoid`, or `tanh`, and the loss function in Eq. 6 remains a similar form. For regression tasks, σ can be omitted and the loss function only contains the inner-class loss with node sets of each class constituted with a single node.

B Related Work

B.1 I/O Unification for Training-free Adaptation

Unified with Language Models. One of the key challenges in developing pre-trained graph foundation models lies in the diverse features and labels. Inspired by the remarkable success of the pre-trained LLMs (OpenAI et al., 2024), researchers have proposed to adopt LMs as I/O modules for graphs (Liu et al., 2023a; Kong et al., 2024; Tang et al., 2024; Zhu et al., 2025). Chen et al. (2024b) investigates the potential of LMs as enhancers for input and predictors for output, separately. ZeroG (Li et al., 2024b) further integrates the two approaches by employing an LM to encode textual descriptions of nodes and classes into unified embeddings. While ZeroG achieves unified input and output for graph pre-training, its applicability is limited to text-attributed graphs (TAGs, *i.e.*, graphs with rich textual features). OFA (Liu et al., 2023a) generalizes TAGs by introducing templates to convert numerical node and edge features into textual descriptions, thus extending its applicability to general graphs. In addition to integrating LMs with GNNs, researchers have also explored pure LLMs in addressing graph-related tasks (Wang et al., 2023; Chen et al., 2024a).

Unified with Specific Design. Distinct from the aforementioned approaches, our work aims to pre-train purely GNN-based models and explore their potential for training-free adaptation. Related efforts in this area include the use of singular value decomposition (SVD) (Sun et al., 2023), Laplacian decomposition (Sun et al., 2025), random projection (Tang et al., 2024), adversarial reprogramming attacks (Jing et al., 2023), and parametric principal component analysis (PCA) (Zhao et al., 2024b)to align different numbers of features. However, these methods couple the parameter values of the input module with specific inputs or fail to unify label spaces for GNNs. A recent approach, GraphAny, attempts to address this challenge by solving the pseudo-inverse of the transformation weight matrix in a linear GNN (Zhao et al., 2024a). Despite the unified input and output for different datasets, GraphAny requires observed labels from test datasets to compute the weight matrix, and is constrained to node-level tasks and linear GNN architectures (Wu et al., 2019). As a result, pre-training graph models with general pure GNN architectures remains an open problem.

B.2 Model Fine-tuning for Few-shot Adaptation

In addition to I/O unification, model fine-tuning has been extensively explored to adapt pre-trained graph models to diverse features and labels (Yu et al., 2024). Approaches like GCC (Qiu et al., 2020) employ full fine-tuning on pre-trained models, which is resource-intensive and prone to overfitting, particularly when downstream datasets involve limited labeled data. To address these limitations, researchers have proposed parameter-efficient graph fine-tuning methods, such as graph adapters (Li et al., 2024a; Gui et al., 2024) and graph prompts (Sun et al., 2023). Graph adapters incorporate additional tunable modules for GNNs, effectively bridging the gap between pre-training and inference domains. Alternatively, graph prompt learning introduces input-specific prompts to modify node features (Sun et al., 2022; Fang et al., 2023; Liu et al., 2023b) or graph structures (Sun et al., 2023; Huang et al., 2023; Zhang et al., 2023b; Tan et al., 2023; Yu et al., 2025), enabling better adaptation to various datasets. Different from model fine-tuning, this paper focuses on directly unifying diverse graph features and labels at the pre-training stage. By exploring this foundational problem, our findings show that the pre-trained GNN models can provide competitive graph operators for further fine-tuning and simplify the hyperparameter tuning process (Sec 4.3).

Table S4: **Original Dataset Split for Training-free Inference.**

	TRAIN	TEST	TOTAL
AMAZONCOMPUTERS	8,252	2,750	13,381
CORA	1,208	1,000	2,708
COAUTHORPHYSICS	20,697	6,898	34,493
ARXIV-YEAR	84,671	42,337	169,343
TWITCH-GAMER	84,057	42,029	168,114
TOLOKERS	5,879	2,940	11,758
CHAMELEON	1,092	456	2,277
ACTOR	3,698	1,520	7,600

Table S5: **Dataset Split for Model Fine-tuning.**

	TRAIN	VALID	TEST	TOTAL
CORA	1,208	500	1,000	2,708
PUBMED	18,217	500	1,000	1,9717
AMAZONCOMPUTERS	8,252	2,379	2,750	1,3381
WIKICS	580	5,274	5,847	1,1701
AMAZON-RATINGS	12,246	6,123	6,123	2,4492
MINESWEEPER	5,000	2,500	2,500	1,0000
TOLOKERS	5,879	2,939	2,940	1,1758

C EXPERIMENTAL SETUP

C.1 SETTINGS

Zero-shot learning with label priors requires test label knowledge. However, label semantics vary across graph datasets, making full coverage of different semantics during pre-training impractical. To align with real-world scenarios, we exclude label knowledge from pre-training. Models do not have access to the label description knowledge during training and target-insensitive prediction. For further evaluation, the prediction results are permuted by aligning pseudo labels with actual labels. Specifically, to ensure fair comparison with LM-based graph models in the zero-shot setting, we employ the embedded label semantics $\mathbf{S}_{\texttt{tgt}}^{\texttt{LM}}$ from language models (Li et al., 2024b) and construct the assignment matrix with $\texttt{f}_{\texttt{out}}(c, s)\mathbf{S}_{\texttt{tgt}}^{\texttt{LM}^\top}$. This scenario does not require any labeled samples from test datasets. To explore the potential of pre-trained pure GNNs on general downstream tasks, observed target labels are only included during inference to match pseudo labels with real labels. Given the observed real labels $\mathbf{C}$, the mapping relations between the pseudo labels and the real labels can be formulated as $\hat{\mathbf{C}}\mathbf{P}(c\mathbf{I} - \mathbf{1}\mathbf{1}^\top) = c\texttt{log}(c\mathbf{C})$. The assignment matrix $\mathbf{P}$ can be solved as the least-squares solution of the linear equation without additional training effort.

C.2 DATASETS

Various real-world datasets are adopted for model pre-training and evaluation. These datasets can be categorized into four domains, including electronic-commerce graphs (e-com.), citation graphs, social graphs, and Wikipedia graphs (wiki). Three levels of average node number per dataset including 1k, 10k, and 100k are incorporated for each domain. The statistics of these datasets are summarized in Tab. S11 and Tab. S12. For the test datasets, we follow the standard split as the supervised learning setting in the original paper.

The e-com. domain consists of one test dataset AmazonComputers (Shchur et al., 2019), and three train datasets AmazonPhoto (Shchur et al., 2019), amazon-ratings (Platonov et al., 2023), and ogbn-products (Hu et al., 2020). All these datasets are collected from Amazon. The citation domain consists of three test datasets Cora (Sen et al., 2008), CoauthorPhysics (Shchur et al., 2019), and arxiv-year (Lim et al., 2021), and four train datasets CiteSeer (Sen et al., 2008), PubMed (Sen et al., 2008), ogbn-arxiv (Hu et al., 2020), and snap-patents (Lim et al., 2021). These datasets are collected from academic graphs, encoding coauthorship and citation relations. The social domain consists of two test datasets twitch-gamer (Lim et al., 2021), tolokers (Platonov et al., 2023), and five train datasets twitch-e (Lim et al., 2021), fb100 (Lim et al., 2021), genius (Lim et al., 2021),

Table S6: **Textual Dataset Statistics.**

	DATASET	USAGE	#NODES	#EDGES	#LABELS
CITATION	ARXIV	TRAIN	169,343	2,315,598	40
	PUBMED	TRAIN	19,717	88,648	3
	DBLP	TEST	14,376	431,326	4
E-COM.	AMAZON-RATINGS	TRAIN	24,492	186,100	5
	BOOKHISTORY	TRAIN	41,551	503,180	12
	BOOKCHILD	TEST	76,875	2,325,044	24
	COMPUTERS	TEST	87,229	1,256,548	10
	PHOTO	TEST	48,362	873,782	12
	SPORTSFIT	TEST	173,055	3,020,134	13
	PRODUCTS	TEST	316,513	19,337,722	39
WIKI.	WIKICS	TEST	11,701	431,726	10
SOCIAL	TOLOKERS	TEST	11,758	1,038,000	2

Facebook (Rozemberczki et al., 2021), and pokec (Lim et al., 2021). These datasets are collected from online social media, encoding social relationships between different users. Specifically, tolokers encapsulates crowdsourcing participation data sourced from the Toloka platform. Edges in tolokers link toloker pairs that have completed the same tasks. This graph indicates certain interests of the participants. Therefore, we classify it in the social domain. The wiki. domain consists of two test datasets CHAMELEON (Pei et al., 2019), ACTOR (Pei et al., 2019), and three train datasets WikiCS (Mernyei & Cangea, 2022), romain-empire (Platonov et al., 2023), and NELL (Carlson et al., 2010). These datasets are collected from Wikipedia. The splits employed in Tab. 1 and Tab. 3 are summarized in Tab. S4 and Tab. S5, respectively.

We also employ textual datasets (Chen et al., 2024c) to compare with LM-based models. The datasets can also be categorized into citation, e-com., social, and Wiki graphs. The citation graphs include arxiv, PubMed, and DBLP. The e-com. graphs include amazon-ratings, bookhistory, bookchild, computers, photo, sportsfit, and products. The Wiki. graph refers to WikiCS, and the social graph refers to tolokers. The statistics of these datasets are summarized in Tab. S6.

C.3 IMPLEMENTATION

Eight GNN methods are employed as the backbone for model pre-training, including GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018a), GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019), MixHop (Abu-El-Haija et al., 2019), GraphGPS (Rampášek et al., 2022), DeepGCN (Li et al., 2019), and $\mathbf{N^2}$ (Sun et al., 2024). The evaluations are conducted on a single NVIDIA GeForce RTX 4090 or a single NVIDIA A100. Models are pre-trained on the `TRAIN` datasets and evaluated on the `TEST` datasets in Tab. S11.

Except for $\mathbf{N^2}$, backbones are implemented with the framework of PyTorch Geometric. The pre-training process is conducted for 5000 epochs with a learning rate fixed at $1e-5$. The supervised result reproducing is conducted for 500 epochs and will be early stopped if there is no further reduction in the validation loss during 200 epochs. The total epoch for model fine-tuning is 1000, with early-stopping for 200 epochs. We adopt Adam (Kingma & Ba, 2015) as optimizer and set weight decay as 1×10^{-6}. The supervised results are reproduced under the same architecture, with grid search performed on the number of layers in $\{2, 3, 5, 10\}$, dropout in $\{0., 0.1, 0.2, 0.3, 0.5\}$, and the number of hidden dimensions in $\{64, 128, 256\}$. The self-supervised models are first pre-trained in the self-supervised setting and then frozen with a trainable linear output for supervised learning. The hyperparameter configuration follows the original implementation. For model fine-tuning, we perform grid search on dropout in $\{0., 0.1, 0.2, 0.3, 0.5\}$ based on the validation results. For the information loss study in Appendix D.7, grid search is performed on the number of layers in $\{2, 3, 5, 10\}$, and the number of hidden dimensions in $\{64, 128\}$. Except for performance comparison with baselines on textual/non-textual datasets and pre-training with scaling parameters, we fix the number of dimensions at 256 for all backbones. The configuration for the number of layers during pre-training for data scaling and domain gap is presented in Tab. S7.

Table S7: **Layer Configuration.**

	#LAYERS
GCN	6
GAT	6
GRAPHSAGE	6
GIN	2
DEEPGCN	1
$\mathbf{N^2}$	6
GRAPHGPS	1
MIXHOP	8

Table S8: **Evaluation Results with Graph-level Datasets (Measured by accuracy: %).** Bold values denote the best results per test dataset. LP, Lap, and Rand denote Label Propagation, Laplacian decomposition, and random projection, respectively.

	ENZYMES	REDDIT-BINARY	PROTEINS	AMAZONCOMPUTERS	CORA	ARXIV-YEAR	TWITCH-GAMER
ORIGINAL SPLIT							
GCN (SUP)	37.41±0.32	**78.40**±0.41	71.70±0.39	91.09±0.13	81.80±0.28	48.03±0.41	59.44±0.18
GRAPHCL	19.77±0.41	71.26±0.32	67.23±1.01	88.67±0.48	61.93±1.47	OOM	OOM
LAP	52.38±0.37	72.04±0.71	70.07±0.52	78.69±0.76	76.80±0.46	40.86±0.64	56.25±0.50
RAND	18.33±0.37	52.20±0.43	59.67±0.73	34.73±0.68	15.50±0.71	37.00±0.76	52.29±0.44
SVD	53.65±0.54	71.08±0.54	70.89±0.69	75.67±0.49	73.50±0.66	40.57±0.55	56.82±0.52
FUG	-	-	-	88.22±0.09	30.70±0.96	42.54±0.30	58.16±0.27
GRAPHANY	-	-	-	82.94±0.82	79.41±0.35	38.36±0.53	59.96±0.02
UNIFIED I/O (NODE)	52.83±1.06	69.65±0.36	70.73±0.26	89.85±0.18	82.32±0.97	42.58±0.17	59.99±0.05
UNIFIED I/O (GRAPH)	**54.88**±0.40	73.09±0.43	**72.45**±0.44	80.29±0.55	80.30±0.70	41.35±0.43	57.73±0.66
1-SHOT FOR TRAINING-FREE INFERENCE							
GRAPHCL	16.71±0.69	52.85±1.53	51.89±3.68	41.39±2.64	27.83±1.20	OOM	OOM
LAP	17.33±2.46	56.88±1.66	55.62±1.91	45.71±2.18	22.80±2.00	28.57±1.90	53.45±1.88
RAND	17.68±2.06	55.36±1.86	49.56±1.56	15.67±1.78	16.20±2.26	26.61±1.93	52.56±2.16
SVD	17.87±1.37	56.80±1.35	55.47±1.60	43.46±1.27	18.90±2.44	24.78±1.37	53.45±1.17
FUG	-	-	-	27.26±0.36	41.83±0.26	27.58±0.11	49.93±0.01
GRAPHANY	-	-	-	62.87±0.29	53.63±1.03	25.03±0.48	49.65±0.48
UNIFIED I/O (NODE)	17.82±1.27	48.21±1.34	52.27±1.75	59.89±0.80	43.94±0.50	33.47±0.14	57.90±0.15
UNIFIED I/O (GRAPH)	**18.08**±1.82	**58.89**±2.40	**56.07**±0.81	60.33±2.16	25.90±0.92	29.96±0.95	54.60±1.47
3-SHOT FOR TRAINING-FREE INFERENCE							
GRAPHCL	18.16±0.88	57.13±1.46	53.73±0.82	55.41±1.41	34.97±0.64	OOM	OOM
LAP	20.42±1.29	58.23±1.23	58.60±1.26	45.35±1.10	34.80±1.39	24.62±1.35	54.51±1.26
RAND	18.62±1.28	56.30±1.16	51.38±1.16	19.02±1.31	30.70±1.37	25.64±1.09	53.04±1.43
SVD	20.67±1.10	58.28±1.16	55.95±1.35	50.76±1.36	35.60±1.26	21.62±1.23	53.26±1.13
FUG	-	-	-	50.59±0.29	47.77±0.29	24.02±0.19	49.83±0.09
GRAPHANY	-	-	-	70.04±1.43	66.32±1.21	24.74±0.34	54.71±0.18
UNIFIED I/O (NODE)	19.00±1.93	49.07±1.68	54.46±1.36	68.33±0.28	49.21±0.91	35.32±0.29	57.64±0.06
UNIFIED I/O (GRAPH)	**22.05**±2.00	**59.32**±0.93	**60.46**±1.57	65.78±2.27	38.90±1.17	23.97±1.48	55.56±1.43

D ADDITIONAL RESULTS

D.1 GRAPH-LEVEL TASKS

As noted in Appendix A, except for the node-level evaluation, unified I/O can be employed for more tasks. To demonstrate this, GNNs are pre-trained on graph-level datasets (COLLAB, IMDB-BINARY, MUTAG, and D&D) (Morris et al., 2020), and evaluated on graph-level tasks (PROTEINS, REDDIT-BINARY, ENZYMES) (Morris et al., 2020), and node-level tasks (AmazonComputers (Shchur et al., 2019), Cora (Watts & Strogatz, 1998), arxiv-year, twitch-gamer (Lim et al., 2021)). Baselines include supervised method GCN (Kipf & Welling, 2017), self-supervised learning method GraphCL (You et al., 2020), parameter-free feature alignment methods SVD (Sun et al., 2023), Laplacian projection (Sun et al., 2025), and random projection (Tang et al., 2024)) combined with our unified output module. We also adopt FUG (Zhao et al., 2024b) and GraphAny (Zhao et al., 2024a) as baselines for node-level tasks. Neither methods support graph tasks and are thus pre-trained on (amazon-ratings, ogbn-arxiv, Facebook, and roman-empire).

Results in Tab. S8 show that pre-training with unified I/O surpasses GraphCL and parameter-free feature alignment methods on both graph-level and node-level tasks. On graph-level tasks, unified I/O also delivers performance comparable to, or better than, supervised GCN. However, transferring between graph-level and node-level tasks introduces a noticeable performance gap in both directions. In particular, for node-level downstream tasks, models pre-trained on graph-level datasets perform competitively on the original split but lag behind under the 1-shot and 3-shot settings when compared with models pre-trained directly on node-level datasets.

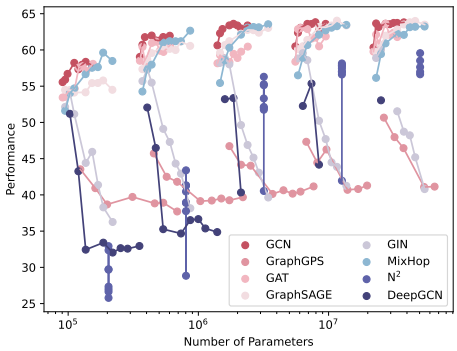
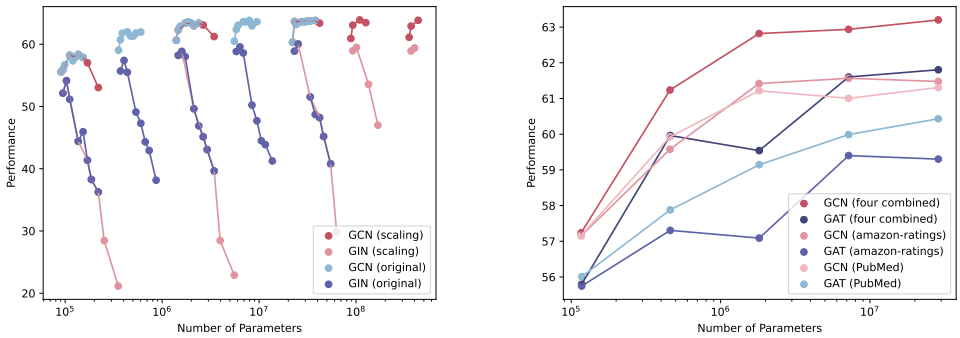


Figure S8: **Pre-training with Scaling Parameters.** The connected dots denote scaling the number of parameters by stacking multiple layers. The adjacent line segments denote scaling by expanding the hidden dimension.



(a) Pre-training with More Parameters (b) Impact of Training Data on Parameter Scaling

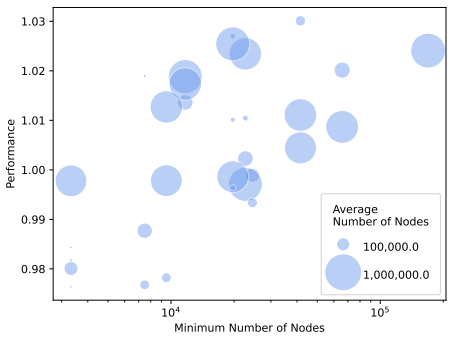
Figure S9: **Over-parameterization in Parameter Scaling.** (a) The connected dots denote scaling the number of parameters by stacking multiple layers. The adjacent line segments denote scaling by expanding the hidden dimension. (b) "Four combined" denotes employing amazon-ratings, ogbn-arxiv, Facebook, and roman-empire for pre-training.

D.2 Scaling Parameters

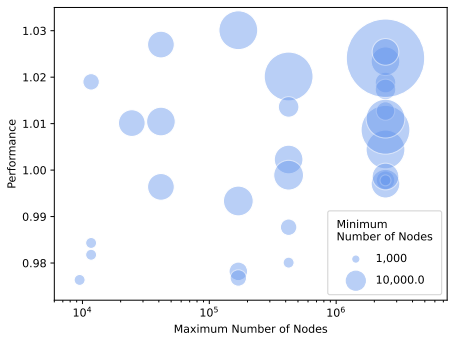
Following the common practice in LLM (OpenAI et al., 2024), model performance is also compared with different numbers of parameters. The experimental settings are the same as Fig. 5, with models pre-trained on (amazon-ratings, ogbn-arxiv, Facebook, roman-empire) and performance averaged across different test graphs. The connected dots denote scaling the number of parameters by stacking multiple layers. Note that $\mathbf{N}^2$ is a recurrent model, where the number of parameters does not change with different layer depths. Results in Fig. S8 exhibit the clear influence driven by layer depth and hidden dimension. Notably, increasing parameter count by adding more layers does not consistently improve performance, while scaling hidden dimensionality is a more stable and beneficial strategy to improve pre-trained GNN models. These observations indicate that model capacity cannot be assessed solely through parameter volume. Instead, layer depth must be chosen appropriately for each architecture, as simply increasing parameters by stacking layers does not always yield better performance.

We further explore the boundary of parameter scaling with more parameters, *i.e.*, hidden dimensions of $\{2048, 4096\}$ and depths of $\{20, 32\}$ layers. Fig. S9(a) shows that further scaling the number of layers (connected dots) causes over-parameterization. This can be alleviated by expanding training datasets, where results in Fig. S9(b) indicate that more training datasets can better support parameter scaling.

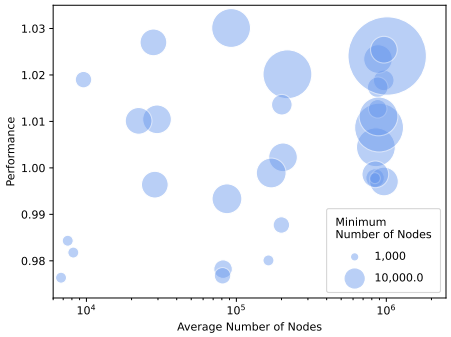
In contrast, increasing the number of hidden dimensions to 4096 does not show over-parameterization. This observation further supports the conclusion that scaling hidden dimensionality is a more stable and beneficial strategy for improving pre-trained GNN models.



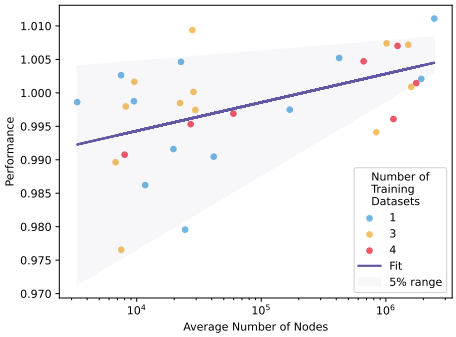
(a) Fix the Minimum, Increase the Average



(b) Fix the Maximum, Increase the Minimum



(c) Fix the Average, Increase the Minimum



(d) Scaling with the Average Number of Nodes

Figure S10: **Study on the Data Scaling Strategy.** (c) Different dot colors indicate the number of datasets employed for pre-training. "5%" denotes envelope-fitting with the top and bottom 5% of the data in each bin.

D.3 Data Scaling Strategy

To explore an effective data scaling strategy, we compare model pre-training with different training data. Specifically, different combinations of three datasets are employed as training data. Model performance is normalized by dividing the median result of the same backbone pre-trained with various datasets on the corresponding test dataset. Fig. S10(a) shows the results of fixing the minimum dataset size and varying the average number of nodes. When the minimum training dataset size remains, increasing the average number of nodes does not consistently improve performance. This indicates that simply adding larger datasets while keeping the smallest size unchanged is not an effective way to expand the training set.

Moreover, Fig. S10(b) and Fig. S10(c) show the results of fixing the maximum dataset size or the average number of nodes and increasing the minimum. In both cases, raising the minimum dataset sizes yields improved performance. Together, these findings suggest that mixing datasets with widely varying node scales is inefficient; datasets should instead be chosen to maintain a similar node scale. Therefore, we compare dataset combinations that are aligned in node scale for the data scaling study in Fig. 6. Additionally, we directly expand the average number of nodes in dataset combinations without considering the smallest dataset size. Model performance is normalized by dividing the median result of the same backbone pre-trained with various datasets on the corresponding test dataset. The averaged results over 1,848 data points across different backbones and test datasets are

presented in Fig. S10(d). Results show that increasing the average number of nodes benefits model performance. In contrast, incorporating more training datasets with various node scales does not necessarily result in superior performance, which verifies the inefficiency of this strategy.

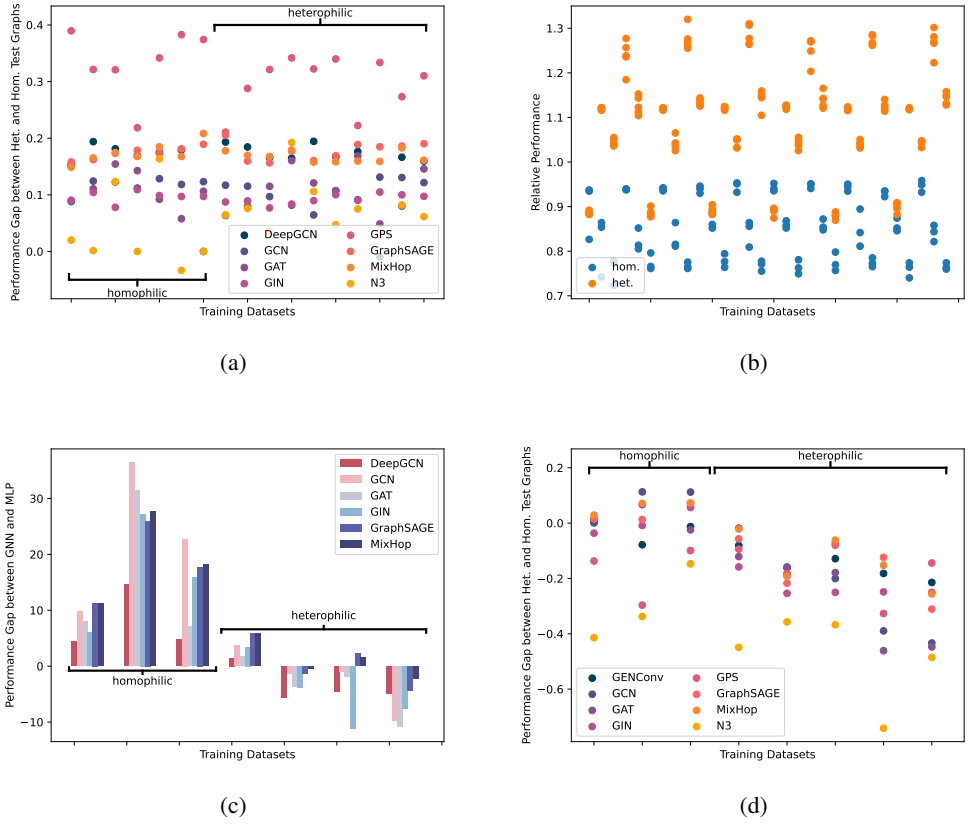


Figure S11: **Adaptation Performance Comparison on Heterophilic and Homophilic graphs.** **(a)** presents the difference between the performance of the same pre-trained model evaluated on heterophilic graphs and homophilic graphs. **(b)** presents the normalized performance of MLP by dividing the state-of-the-art supervised results of classic GNN models on the corresponding test datasets. **(c)** presents the difference between the absolute performance of pre-trained GNNs and MLP. **(d)** presents the difference between the performance of pre-trained GNNs and MLP normalized by their supervised counterpart.

D.4 Inference with Different Homophily-Heterophily

Comparison with the same pre-training dataset. Sec. 4.2 examines model adaptation using different pre-training datasets on the same test dataset, showing that training on datasets with the same homophily–heterophily characteristics as the test dataset generally yields better results. In this section, we shift the focus to comparing model performance across different test datasets under the same pre-training setting. However, direct comparison is challenging because the inherent difficulty of each test dataset varies, resulting in different absolute performance values. To address this issue, we normalize the results by dividing them by the corresponding reproduced supervised results. A higher normalized value indicates that the pre-trained model adapts more effectively to the corresponding downstream task. Fig. S11(a) presents the difference between the averaged performance of the same pre-trained model on heterophilic test datasets and homophilic test datasets ($\texttt{Metric}_{\texttt{het}} - \texttt{Metric}_{\texttt{hom}}$). We can see that models pre-trained on either homophilic or heterophilic graphs gain a positive performance gap, indicating better results on heterophilic graphs than homophilic graphs. To further study this phenomenon, we contrast pre-trained MLPs with pre-trained GNNs to evaluate the requirement of

homophilic and heterophilic graphs during model adaptation, and thus derive the root cause of the phenomenon for Fig. S11(a).

Requirement of heterophilic graphs. We compare the performance of pre-trained MLP with supervised GNNs. Supervised GNNs are directly optimized to meet task-specific requirements on downstream datasets. Comparing them with pre-trained models assesses whether the latter also fulfill these requirements. Fig. S11(b) presents the normalized performance of MLP by dividing the state-of-the-art supervised results of classic GNN models on the corresponding test datasets. MLP achieves more comparable performance on heterophilic test graphs compared to supervised results. This indicates that node features better benefit the adaptation to heterophilic graphs. In contrast, the requirement of the adaptation to homophilic graphs cannot be well satisfied by MLP and leads to inferior normalized performance.

Requirement of homophilic graphs. We compare the performance of pre-trained MLPs and GNNs. MLPs that only employ node features serve as a baseline for adapting to homophilic graphs. Comparing more complex models with MLP highlights the further requirement of homophilic graphs. Fig. S11(c) presents the performance gap between pre-trained GNNs and MLPs. We can see that GNNs gain better performance on homophilic graphs and similar performance on heterophilic graphs compared to MLP. This indicates that capturing graph structures contributes to the adaptation to homophilic graphs.

Transferability of the learned knowledge. Given the requirement of capturing node features to adapt to heterophilic graphs and capturing structures for homophilic graphs, we further analyze the difficulty of transferring these learned patterns. Specifically, supervised models transfer knowledge within the same dataset, while pre-trained models transfer across datasets. Therefore, comparing the performance of pre-trained models with their supervised counterparts shows the transferability of knowledge learned during pre-training, where GNNs correspond to the structural knowledge and MLP corresponds to the node feature knowledge. Fig. S11(d) presents the normalized performance gap between pre-trained GNNs and MLP, where pre-training results are divided by the supervised results of the same model on the same datasets. Results show that the performance gap between normalized pre-trained GNNs and MLP is generally negative, where the pre-trained MLP is more comparable to its supervised counterpart. This suggests that the node feature knowledge is consistently transferable within and across datasets. Conversely, structural knowledge transfers well within the same dataset but fails to generalize across different datasets. As a result, pre-trained GNNs with only transferable node feature knowledge cannot satisfy the requirement of the homophilic graphs and thus achieve better normalized performance on heterophilic graphs.

Based on the above conclusions, the phenomenon in Fig. S11(a) can be attributed to the inherent differences between homophilic and heterophilic graphs. Adapting to heterophilic graphs mostly requires the capturing of node features, while adapting to homophilic graphs requires models to adhere closely to the input graph structures. However, structural knowledge fails to transfer across different datasets compared to the better transferability of the node feature knowledge, resulting in consistently better performance when adapting to heterophilic graphs.

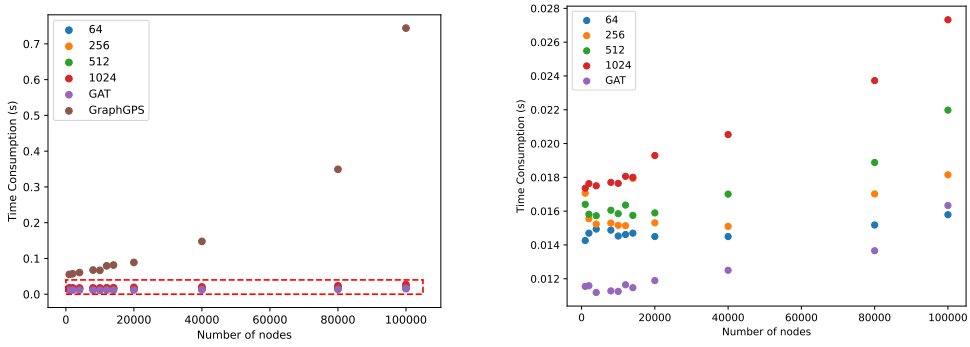
D.5 Model Fine-tuning

Our unified I/O modules enable seamless adaptation of pure GNN architectures across diverse datasets. To further evaluate the effectiveness of the pre-trained GNN operators, we fine-tune the models pre-trained on amazon-ratings, ogbn-arxiv, Facebook, and roman-empire. The internal GNN module $\texttt{F}_\texttt{g}$ within the pre-trained models is frozen during the fine-tuning. The unification-oriented I/O function $\texttt{f}_{\texttt{in}}(\cdot)$ in Eq. 4 and $\texttt{f}_{\texttt{out}}(\cdot)$ in Eq. 5 are replaced with learnable parameters. We implemented GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018a), GIN (Xu et al., 2019), GraphSAGE (Hamilton et al., 2017), and GraphGPS (Rampášek et al., 2022) in the supervised-learning setting and DGI (Veličković et al., 2018b), GRACE (Zhu et al., 2020), GraphACL (Xiao et al., 2023), GraphCL (You et al., 2020), SimGRACE (Xia et al., 2022), MaskGAE (Li et al., 2023), GraphMAE2 (Hou et al., 2023) in the self-supervised setting. The fine-tuning results with different backbones on downstream graphs are summarized in Tab. S9. The pre-trained operators achieve superior performance to supervised methods and self-supervised methods. Notably, the pre-trained operators require minimal hyperparameter tuning, with only dropout adjusted during fine-tuning. This

Table S9: **Evaluation Results of Fine-tuning the I/O Modules in the Pre-trained GNNs (Measured by accuracy except ROC AUC for tolokers: %).** Bold values denote the best results per test dataset.

	CORA	PUBMED	AMAZON COMPUTERS	WIKICS	AMAZON -RATINGS	MINESWEEPER	TOLOKERS
SELF-SUPERVISED							
DGI	84.00±0.28	83.73±0.41	82.11±0.16	75.05±0.39	40.80±0.74	88.45±0.38	77.73±0.14
GRACE	84.30±0.57	85.81±0.18	89.67±0.36	75.80±0.56	42.19±0.12	86.15±0.45	75.06±0.14
GRAPHACL	75.00±0.75	82.93±0.17	80.58±0.15	68.00±0.75	40.65±0.14	87.23±0.11	77.68±0.25
GRAPHCL	63.33±1.76	63.53±1.08	84.57±0.34	76.32±0.18	42.35±0.37	79.85±0.12	80.03±0.12
MASKGAE	75.13±1.78	75.27±1.03	92.15±0.05	78.25±0.20	43.54±0.30	84.33±0.16	81.13±0.34
SIMGRACE	67.07±0.82	77.63±0.82	87.44±0.18	78.75±0.28	43.46±0.23	84.23±0.16	80.29±0.30
GRAPHMAE2	79.50±0.51	67.07±0.91	91.03±0.19	76.24±0.11	40.95±0.71	80.16±0.10	80.17±0.07
SUPERVISED							
GRAPHSAGE	78.83±0.50	88.11±0.05	91.09±0.02	78.13±0.15	45.71±0.38	90.55±0.10	83.06±0.59
GAT	77.51±2.35	85.30±0.15	89.78±0.02	76.35±0.80	44.54±0.52	82.07±1.17	77.37±0.28
GIN	77.36±0.15	85.13±0.55	90.51±0.80	74.02±0.62	46.33±0.11	74.93±0.58	60.93±2.25
GCN	80.35±0.25	85.44±0.50	90.66±0.13	78.55±0.01	46.71±0.25	76.43±1.05	77.79±0.12
GRAPHGPS	58.61±0.05	85.21±0.30	88.87±0.20	75.18±0.04	47.85±0.29	89.64±0.24	79.82±0.06
PRE-TRAINED AND FINE-TUNED							
GCN	84.32±0.09	85.28±0.21	91.02±0.01	78.42±0.07	46.17±0.09	69.10±0.11	69.50±0.69
GAT	78.01±0.87	84.47±0.33	89.74±0.42	77.96±0.13	47.13±0.45	71.03±0.72	75.67±0.73
GIN	79.31±0.82	85.61±0.38	87.89±0.43	73.49±0.14	49.93±0.22	77.62±0.28	66.83±0.56
GRAPHGPS	50.62±0.42	86.79±0.71	85.93±0.23	73.41±0.74	43.74±0.26	88.68±0.29	80.41±0.53
GRAPHSAGE	83.92±0.43	86.37±0.23	91.24±0.17	**78.98**±0.12	48.85±0.82	**91.39**±0.15	**83.29**±0.13
MIXHOP	**84.63**±0.12	**88.91**±0.45	90.33±0.30	78.93±0.17	**51.59**±0.05	90.77±0.18	83.03±0.13
N^2	81.50±0.33	88.32±0.32	**92.33**±0.25	76.60±0.25	49.85±0.31	90.31±0.31	81.51±0.36
DEEPGCN	74.40±0.33	88.50±0.32	91.02±0.25	74.67±0.25	50.76±0.46	88.00±0.28	79.86±0.36

significantly simplifies the hyperparameter tuning process, enabling efficient adaptation of pre-trained GNNs to various graphs with promising performance.



(a) Different graph sizes (b) Different numbers of input features

Figure S12: **Time comparison.** (b) zooms in on the red block in (a).

D.6 COMPLEXITY ANALYSIS

The space complexity of the unified I/O is $O(n)$, with $d_{\text{in}}, d, s, c \ll n$. To evaluate the time cost of different modules, We construct synthetic graphs. Nodes in the synthetic graphs have an average degree of 20. The largest number of edges is 100,000. The number of classes is 10, which is close to the common configuration of the real-world datasets in Fig. 1. The number of GNN layers is fixed to 3. The results are presented in Fig. S12. As the number of input features increases, the time cost of the unified I/O module increases but is constantly less than the cost of the GraphGPS module. The I/O time cost also scales linearly with the graph size. This demonstrates the efficiency of the proposed unified I/O modules.

Table S10: **Information Loss Study.** "Ori" and "Uni" denote the supervised learning results with traditional I/O modules and our unified I/O modules, respectively.

	AMAZON-RATINGS	ARXIV-YEAR	COAUTHORCS	COAUTHORPHYSICS
GCN (ORI.)	48.70	46.02	92.92	96.18
GCN (UNI.)	47.12	45.19	92.77	96.38
GAT (ORI.)	52.70	46.05	93.61	96.17
GAT (UNI.)	46.19	46.26	88.38	94.16
GRAPHSAGE (ORI.)	53.63	43.76	93.91	96.49
GRAPHSAGE (UNI.)	45.45	44.76	95.17	97.06

D.7 EFFECTIVENESS OF THE UNIFIED I/O

Information Loss. Unified I/O decouples the learnable parameters from the numbers and semantics of dimensions for the feature and label space. To evaluate whether this leads to information loss, we conduct supervised training with our unified I/O on GCN, GAT, and GraphSAGE. As presented in Tab. S10, our unified I/O modules do not cause severe information loss. They even enable the backbone GNN methods to achieve better performance than that of their vanilla architectures for GAT and GraphSAGE on arxiv-year, GraphSAGE on CoauthorCS, GCN and GraphSAGE on CoauthorPhysics. This demonstrates the effectiveness of our unified I/O in learning input and output mappings.

(a) Given Semantics (b) Learned Semantics

Figure S13: **t-SNE Results for Semantics Alignment Comparison.** Each dot represents a different feature channel, with dot colors representing different datasets.

Feature Semantics. To explore whether our parametric function $\mathtt{f}_{\mathtt{in}}(\cdot)$ empowers feature semantics unification across different inputs, we use t-SNE to visualize feature semantics from (Cora, CiteSeer, Photo, Computers, ogbn-products, and WikiCS) (Chen et al., 2024c). 40 feature channels are sampled for each dataset. Each dot represents a different feature channel, with dot colors representing different datasets. Fig. S13(a) shows the t-SNE result of the original feature semantics, where features from different datasets form clearly separated clusters rather than merging into a shared global structure. This pronounced dataset separation indicates that the representations are not well aligned within a common semantic space. Conversely, in Fig. S13(b), feature semantics from different datasets learned by unified I/O form two major clusters, indicating that the learned representations are unified into two common semantic subspaces. Within each cluster, channels from different datasets do not fully mix but show partial segregation, suggesting that the semantic representations still preserve dataset-specific characteristics. All these results demonstrate the effectiveness of our proposed method in modeling unified feature semantics.

E Proof and Derivation

E.1 Mapping with Dimension Relations

Theorem 3.1 (Mapping with Dimension Relations). *Given any linear mapping $\mathbf{W} \in \mathbb{R}^{d_{\texttt{src}} \times d_{\texttt{tgt}}}$ and $s \in \mathbb{N}^+$, there always exist two representation matrices $\mathbf{S}_{\texttt{src}} \in \mathbb{R}^{d_{\texttt{src}} \times s}$ and $\mathbf{S}_{\texttt{tgt}} \in \mathbb{R}^{d_{\texttt{tgt}} \times s}$, such that $\mathbf{W} = \psi(\mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}})$, where $\psi(\cdot, \cdot)$ is a bilinear composition function.*

Definition E.1 (Bilinear Composition Function)**.** *Let $\mathbf{X} \in \mathbb{R}^{m \times d_1}$ and $\mathbf{Y} \in \mathbb{R}^{n \times d_2}$ be two input matrices, and let $\mathbf{U} \in \mathbb{R}^{d_1 \times d_2}$ be a learnable parameter matrix. A bilinear composition function $\psi : \mathbb{R}^{m \times d_1} \times \mathbb{R}^{n \times d_2} \to \mathbb{R}^{m \times n}$ is defined as*

$$\psi(\mathbf{X}, \mathbf{Y}) = \mathbf{X}\mathbf{U}\mathbf{Y}^\top,$$

which computes the bilinear form between each pair of row vectors from $\mathbf{X}$ and $\mathbf{Y}$. This function is linear with respect to either argument when the other is fixed, but not jointly linear.

We now provide the proof for Theorem 3.1:

Proof. Let $r = \texttt{rank}(\mathbf{W}) \leq \min(d_{\texttt{src}}, d_{\texttt{tgt}})$. By the full-rank factorization theorem (Meyer, 2023), there exist matrices $\mathbf{A} \in \mathbb{R}^{d_{\texttt{src}} \times r}$ and $\mathbf{B} \in \mathbb{R}^{d_{\texttt{tgt}} \times r}$ such that

$$\mathbf{W} = \mathbf{A}\mathbf{B}^\top.$$

Let $s = r$, $\mathbf{S}_{\texttt{src}} := \mathbf{A}$, $\mathbf{S}_{\texttt{tgt}} := \mathbf{B}$, and $\mathbf{U}$ be the identity matrix $\mathbf{I}_r$. Then

$$\mathbf{S}_{\texttt{src}}\mathbf{U}\mathbf{S}_{\texttt{tgt}}^\top = \mathbf{A}\mathbf{I}_r\mathbf{B}^\top = \mathbf{A}\mathbf{B}^\top = \mathbf{W}.$$

Thus, such $\mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}}, \mathbf{U}$ always exist.

More generally, for any $s \geq r$, we can embed $\mathbf{A}$ and $\mathbf{B}$ into higher-dimensional matrices by padding zeros and set $\mathbf{U}$ as a diagonal matrix with the first r entries as 1 and others as 0. Hence, the bilinear form $\psi(\mathbf{S}_{\texttt{src}}, \mathbf{S}_{\texttt{tgt}}) = \mathbf{S}_{\texttt{src}}\mathbf{U}\mathbf{S}_{\texttt{tgt}}^\top$ is expressive enough to represent any linear mapping $\mathbf{W}$. □

E.2 Set Learning for the Unified Input Module

To decouple the parameters from the number of source dimensions, feature semantics $\mathbf{S}_{\texttt{src}}^{(\texttt{in})}$ is formulated as a parametric function $\texttt{f}_{\texttt{in}}(\mathbf{X}; \mathcal{W}_{\texttt{in}})$. The function $\texttt{f}_{\texttt{in}}(\cdot)$ is subject to two conditions: (1) Permutation invariance to the order of input nodes and equivariance to that of source dimensions; (2) Size independence of the parameter set $\mathcal{W}_{\texttt{in}}$ to the values of n and $d_{\texttt{in}}$. Given the absence of topological structures and the permutation condition (Cond 1) for $\texttt{f}_{\texttt{in}}(\cdot)$, the input features can be modeled as a set of channels $\{\mathbf{X}_{\cdot,j}\}$, where each channel corresponds to a set of nodes $\{\mathbf{X}_{i,j}\}$. As a result, $\texttt{f}_{\texttt{in}}(\cdot)$ is transformed into a set-learning problem at both the channel level and the node level, $\texttt{f}_{\texttt{in}} = \texttt{f}_{\texttt{in}}^{\texttt{cha}} \circ \texttt{f}_{\texttt{in}}^{\texttt{nod}}$.

Channel-level Set Learning. Based on the universal functions on set (Zaheer et al., 2017), $\texttt{f}_{\texttt{in}}(\cdot)$ is a permutation-equivariant set function at the channel level and can be decomposed as

$$\texttt{f}_{\texttt{in}}(\mathbf{X}) = \texttt{f}_{\texttt{in}}^{\texttt{cha}}\left(\texttt{f}_{\texttt{in}}^{\texttt{nod}}(\mathbf{X})\right) = \sigma\left[\Theta\texttt{f}_{\texttt{in}}^{\texttt{nod}}(\mathbf{X})\right], \tag{S8}$$

where σ can be any nonlinear function, $\Theta \in \mathbb{R}^{d_{\texttt{in}} \times d_{\texttt{in}}}$ denotes the channel mixer. To enable the scalability of the input module for input features with a large number of channels (*e.g.*, NELL $d_{\texttt{in}} = 61,278$, CoauthorPhysics $d_{\texttt{in}} = 8,415$, and CoauthorCS $d_{\texttt{in}} = 6,805$), we follow the linear attention (Katharopoulos et al., 2020) to construct Θ as

$$\Theta = \frac{d_{\texttt{in}}\mathbf{X}^\top\mathbf{X}}{\mathbf{X}^\top\mathbf{X}\mathbf{1}}, \tag{S9}$$

where $\mathbf{1}$ denotes the all-one vector.

Node-level Set Learning. Due to the size independence and the permutation invariance conditions for $\texttt{f}_{\texttt{in}}(\cdot)$, $\texttt{f}_{\texttt{in}}^{\texttt{nod}}(\cdot)$ in Eq. S8 can be formulated as a permutation-invariant set function at the node level. Given $\texttt{f}_{\texttt{in}} : \mathbb{R}^{n \times d_{\texttt{in}}} \mapsto \mathbb{R}^{d_{\texttt{in}} \times s}$, our ultimate target is to model a number of s representations

for each channel. Therefore, we apply s set functions to $\{\mathbf{X}_{i,j}\}$. Each function can be decomposed following the universal set function (Zaheer et al., 2017) as

$$\mathtt{f}_{\mathtt{in}}^{\mathtt{nod}}(\mathbf{X}) = \left[\mathtt{f}_{\mathtt{in}}^{\mathtt{nod},1}(\mathbf{X})||\cdots||\mathtt{f}_{\mathtt{in}}^{\mathtt{nod},s}(\mathbf{X})\right], \quad \mathtt{f}_{\mathtt{in}}^{\mathtt{nod},k}(\mathbf{X}) = \phi^k\left(\sum_{i\in[1,n]}\rho^k(\mathbf{X}^\top)_{\cdot,i}\right), \tag{S10}$$

where the parameters in both ϕ^k and ρ^k are shared for each element in $\mathbf{X}$ to ensure the size independence condition (Cond 2). We share the function ρ^k for all $k \in [1, s]$ as ρ and implement ϕ^k as a parameter-weighting function to keep simplicity. As a result, Eq. S8-Eq. S10 can be complied as

$$\mathtt{f}_{\mathtt{in}}(\mathbf{X}) = \sigma\left[\frac{d_{\mathtt{in}}\mathbf{X}^\top\mathbf{X}}{\mathbf{X}^\top\mathbf{X}\mathbf{1}}\rho(\mathbf{X}^\top)\mathbf{1}\alpha^\top\right], \tag{S11}$$

where $\alpha = \{\alpha_k\} \in \mathbb{R}^{s\times 1}$ denotes the parameter vector with $k \in [1, s]$ to implement ϕ^k, $\mathbf{1}$ denotes the all-one vector to implement the summation in Eq. S10.

Although Zaheer et al. also provides an implementation named Deep Sets based on the universal set function, our input module differs from this specific implementation in several key aspects. Specifically, graph learning requires permutation invariance over nodes, which constitute the set representations in our case. This demands the input module to decouple parameters from the representation dimensionality. In contrast, Deep Sets are not faced with such a condition. We further decompose the set-learning task into a bi-level formulation, while Deep Sets addresses the original single-level formulation. Moreover, our input module employs a linear-attention-like set mixer in Eq. S9, while Deep Sets applies sum or max pooling to mix sets.

E.3 Permutation Invariance of the Unified Input Module

Theorem E.2 (Permutation Invariance of the Unified Input Module)**.** *Let $\mathbf{P} \in \mathbb{R}^{d_{\mathtt{in}}\times d_{\mathtt{in}}}$ be any permutation matrix. Then the source-adaptive input module $\mathtt{F}_{\mathtt{in}}$ is permutation invariant, such that $\mathtt{F}_{\mathtt{in}}(\mathbf{XP}) = \mathtt{F}_{\mathtt{in}}(\mathbf{X})$.*

Proof. Let $\mathbf{P} \in \mathbb{R}^{d_{\mathtt{in}}\times d_{\mathtt{in}}}$ be a permutation matrix. Consider the input module $\mathtt{F}_{\mathtt{in}}(\mathbf{X}) = \sigma\left[\mathbf{X}\mathtt{f}_{\mathtt{in}}(\mathbf{X})\mathbf{S}_{\mathtt{tgt}}^{(\mathtt{in})\top}\right]$, where $\sigma(\cdot)$ is applied element-wise. Applying the permutation giving rise to $\mathtt{F}_{\mathtt{in}}(\mathbf{XP}) = \sigma\left[\mathbf{XP}\mathtt{f}_{\mathtt{in}}(\mathbf{XP})\mathbf{S}_{\mathtt{tgt}}^{(\mathtt{in})\top}\right]$. Specifically, let $\mathbf{r} = \mathbf{X}^\top\mathbf{X}\mathbf{1}/\sqrt{n}$, $\bar{\mathbf{x}} = \mathbf{X}\mathbf{1}_{d_{\mathtt{in}}}/d_{\mathtt{in}}$, $\mathtt{f}_{\mathtt{in}}(\mathbf{XP})$ can be formulated as

$$\begin{aligned}\mathtt{f}_{\mathtt{in}}(\mathbf{XP}) &= \sigma\left(\frac{(\mathbf{XP})^\top\mathbf{XPP}^\top\mathbf{r}\alpha^\top}{(\mathbf{XP})^\top\bar{\mathbf{x}}}\right)\\ &= \sigma\left(\frac{(\mathbf{XP})^\top\mathbf{X}\mathbf{r}\alpha^\top}{(\mathbf{XP})^\top\bar{\mathbf{x}}}\right).\end{aligned}$$

Here, both the division and $\sigma(\cdot)$ are applied element-wise and invariant to consistent column permutation. Therefore, applying the permutation before or after the element-wise operations yields the same result, giving

$$\begin{aligned}\mathtt{f}_{\mathtt{in}}(\mathbf{XP}) &= \sigma\left(\mathbf{P}^\top\frac{\mathbf{X}^\top\mathbf{X}\mathbf{r}\alpha^\top}{\mathbf{X}^\top\bar{\mathbf{x}}}\right)\\ &= \mathbf{P}^\top\sigma\left(\frac{\mathbf{X}^\top\mathbf{X}\mathbf{r}\alpha^\top}{\mathbf{X}^\top\bar{\mathbf{x}}}\right)\\ &= \mathbf{P}^\top\mathtt{f}_{\mathtt{in}}(\mathbf{X}).\end{aligned}$$

Substituting $\mathtt{f}_{\mathtt{in}}(\mathbf{XP})$ into $\mathtt{F}_{\mathtt{in}}(\mathbf{XP})$, we have

$$\begin{aligned}\mathtt{F}_{\mathtt{in}}(\mathbf{XP}) &= \sigma\left[\mathbf{XPP}^\top\mathtt{f}_{\mathtt{in}}(\mathbf{X})\mathbf{S}_{\mathtt{tgt}}^{(\mathtt{in})\top}\right]\\ &= \mathtt{F}_{\mathtt{in}}(\mathbf{X}).\end{aligned}$$

which completes the proof. □

E.4 Pseudo Label Assignment

Given the observed labels $\mathbf{C}$, the mapping relations between the pseudo labels and the observed labels can be formulated as

$$\texttt{softmax}(\hat{\mathbf{C}}\mathbf{P}) = \frac{\texttt{exp}(\hat{\mathbf{C}}\mathbf{P})}{\texttt{exp}(\hat{\mathbf{C}}\mathbf{P})\mathbf{1}\mathbf{1}^\top} = \mathbf{C}, \tag{S12}$$

where $\mathbf{P} \in \mathbb{R}^{c\times c}$ denotes the assignment matrix, $\mathbf{1}$ denotes the all-one vector. For a set of values $\{x_i\}, i \in [1, c]$, the first-order Taylor expansion of $\texttt{exp}(x_i)$ around $\bar{x} = \sum_i x_i/c$ is

$$\texttt{exp}(x_i) \approx \texttt{exp}(\bar{x}) \cdot (1 + x_i - \bar{x}). \tag{S13}$$

As a result, the summation of $\texttt{exp}(x_i)$ can be approximated as

$$\sum_i \texttt{exp}(x_i) \approx \texttt{exp}(\bar{x}) \sum_i (1 + x_i - \bar{x}) = n\texttt{exp}(\bar{x}). \tag{S14}$$

Substituting Eq. S14 in Eq. S12 yields

$$\begin{aligned} \frac{\texttt{exp}(\hat{\mathbf{C}}\mathbf{P})}{\texttt{exp}(\frac{1}{c}\hat{\mathbf{C}}\mathbf{P}\mathbf{1}\mathbf{1}^\top)} &= c\mathbf{C} \\ \texttt{exp}(\hat{\mathbf{C}}\mathbf{P} - \frac{1}{c}\hat{\mathbf{C}}\mathbf{P}\mathbf{1}\mathbf{1}^\top) &= c\mathbf{C} \\ \texttt{exp}[\hat{\mathbf{C}}\mathbf{P}(\mathbf{I} - \frac{1}{c}\mathbf{1}\mathbf{1}^\top)] &= c\mathbf{C} \\ \hat{\mathbf{C}}\mathbf{P}(c\mathbf{I} - \mathbf{1}\mathbf{1}^\top) &= c\texttt{log}(c\mathbf{C}). \end{aligned} \tag{S15}$$

F LLM Usage

In preparing this manuscript, we employed a large language model (LLM) exclusively for surface-level language refinement, such as grammar correction and improving clarity of expression. The LLM did not contribute to method ideation and experimental study.

G Limitation

In this paper, we explore the training-free adapting capability of the pre-trained models with pure GNNs. However, although the proposed method can be employed for any graph learning task (Appendix A), the empirical evaluation in this paper is limited to node classification. Further study on graph-level and edge-level tasks is left for future work.

Table S11: **Dataset Statistics.** "✓" marks the training datasets as different scales, where "S", "M", "L" denote small-scale, middle-scale, and large-scale datasets with numbers of nodes around 1k, 10k, and 100k, respectively.

	Dataset	Usage	Type	#Nodes	#Features	#Labels	E-com. ONLY			Citation ONLY			Social ONLY			Wiki. ONLY		Wiki. ONLY
							S	M	L	S	M	L	S	M	L	S	M	L
E-com.	AmazonComputers	TEST	HOM.	13,381	767	10												
	AmazonPhoto	TRAIN	HOM.	7,487	745	8	✓											
	Amazon-ratings	TRAIN	HET.	24,492	300	5		✓										
	ogbn-products	TRAIN	HOM.	2,449,029	100	47			✓									
Citation	Cora	TEST	HOM.	2,708	1,433	7												
	CoauthorPhysics	TEST	HOM.	34,493	8,415	5												
	arxiv-year	TEST	HET.	169,343	128	5												
	CiteSeer	TRAIN	HOM.	3,327	3,703	6				✓								
	PubMed	TRAIN	HOM.	19,717	500	3					✓							
	ogbn-arxiv	TRAIN	HOM.	169,343	128	49						✓						
	snap-patents	TRAIN	HET.	2,923,922	269	5												
Social	twitch-gamer	TEST	HET.	168,114	7	2												
	tolokers	TEST	HET.	11,758	10	2												
	twitch-e	TRAIN	HET.	9,498	128	2							✓					
	fb100	TRAIN	HET.	41,554	5	2								✓				
	genius	TRAIN	HET.	421,961	12	2									✓			
	facebook	TRAIN	HOM.	22,470	128	4												
	pokec	TRAIN	HET.	1,632,803	65	2												
Wiki.	chameleon	TEST	HET.	2,277	2,325	5												
	actor	TEST	HET.	7,600	932	5												
	wikics	TRAIN	HOM.	11,701	300	10										✓		
	roman-empire	TRAIN	HET.	22,662	300	18											✓	
	NELL	TRAIN	HET.	65,755	61,278	186												✓

Table S12: **Dataset Combinations for Pre-training.** "✓" marks the selected datasets for different setups. "S", "M", "L" denote small-scale, middle-scale, and large-scale datasets with numbers of nodes around 1k, 10k, and 100k, respectively. "No XX" denotes pre-training on datasets from three domains and adapting to the remaining one for the domain gap experiment.

	Dataset	Usage	Type	#nodes	#features	#labels	No E-com.			No Citation			No Social			No Wiki.			All			Het.	Hom.	Mix.	
							S	M	L	S	M	L	S	M	L	S	M	L	S	M	L				
E-com.	AmazonComputers	test	hom.	13,381	767	10																			
	AmazonPhoto	train	hom.	7,487	745	8				✓			✓			✓			✓						
	amazon-ratings	train	het.	24,492	300	5					✓			✓			✓			✓		✓			✓
	ogbn-products	train	hom.	2,449,029	100	47						✓			✓			✓			✓		✓	✓	
Citation	Cora	test	hom.	2,708	1,433	7																			
	CoauthorPhysics	test	hom.	34,493	8,415	5																			
	arxiv-year	test	het.	169,343	128	5																			
	CiteSeer	train	hom.	3,327	3,703	6	✓						✓			✓			✓						
	PubMed	train	hom.	19,717	500	3		✓						✓			✓			✓					
	ogbn-arxiv	train	hom.	169,343	128	40			✓						✓			✓			✓		✓		✓
	snap-patents	train	het.	2,923,922	269	5																✓		✓	
Social	twitch-gamer	test	het.	168,114	7	2																			
	Tolokers	test	het.	11,758	10	2																			
	twitch-e	train	het.	9,498	128	2	✓			✓						✓			✓						
	fb100	train	het.	41,554	5	2		✓			✓						✓			✓					
	genius	train	het.	421,961	12	2			✓			✓						✓			✓				
	facebook	train	hom.	22,470	128	4																	✓		✓
	pokec	train	het.	1,632,803	65	2																✓		✓	
Wiki.	chameleon	test	het.	2,277	2,325	5																			
	actor	test	het.	7,600	932	5																			
	wikics	train	hom.	11,701	300	10	✓			✓			✓						✓				✓	✓	
	roman-empire	train	het.	22,662	300	18		✓			✓			✓						✓		✓			✓
	NELL	train	het.	65,755	61,278	186			✓			✓			✓						✓				