GRAPH IN GRAPH NEURAL NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Most existing Graph Neural Networks (GNNs) frequently suffer from two limitations: (i) they can only process graphs whose vertices are represented by vectors or single values; and (ii) they assume each input graph is independent from others during the propagation. In this paper, we propose **the first GNN model (called Graph in Graph Neural Network (GIG)) that can process graphs whose vertices are also represented by graphs**. Considering that the relationship between different graphs may contain crucial task-related cues, we further propose a GIG graph relationship modelling (GRM) strategy that integrates multiple target graph samples as a global graph, each of whose vertex describes a target graph sample. We then applies the GIG model to jointly process the combined graph samples (i.e., the global graph), where additional task-specific relationship cues among graph samples can be extracted in an end-to-end manner. The experimental results show that the proposed GIG model and the GRM strategy generalize well on various graph analysis tasks, providing new state-of-the-art results on five out of seven benchmark graph datasets. Importantly, not only its vertex/edge updating functions are flexible to be customized from different existing GNNs but also it is robust to different settings. Our anonymous code and model settings are provided in the supplementary material and appendix for reproducibly purpose.

1 INTRODUCTION

Recent advances in Graph Neural Networks (GNNs) allow task-specific features to be effectively extracted from non-Euclidean graph samples for various graph analysis tasks. While most early GNNs (Perona & Malik, 1990; Battaglia et al., 2016; Marcheggiani & Titov, 2017; Veličković et al., 2017) were built upon anisotropic operations and can only update vertex features based on their first-order neighbours, other approaches (Scarselli et al., 2008; Bruna et al., 2013; Defferrard et al., 2016; Sukhbaatar et al., 2016; Kipf & Welling, 2016; Veličković et al., 2018) extended Convolution Neural Networks (CNNs) to the graph domain, allowing information to be exchanged between vertices and their higher-order neighbours. In the past three years, more advanced GNNs have been proposed to allow them to be able to not only process graphs that have different typologies (heterogeneous graphs) (e.g., Heterogeneous Graph Neural Network (Zhang et al., 2019a), Heterogeneous Graph Attention Network (Wang et al., 2019b), etc.) but also graphs whose edges are represented by multi-dimensional vectors (e.g., EGNNs (Gong & Cheng, 2019), ME-GCN Wang et al. (2022), and MDE-GNN (Xiong et al., 2021), etc.). Moreover, a recent graph benchmark study (Dwivedi et al., 2020) utilized the advantages of such advanced GNNs by integrating both heterogeneous graph and multi-dimensional edge feature processing mechanisms into multiple widely-used GNNs (e.g., Graph Attention Network (GAT) (Veličković et al., 2017), Gated Graph ConvNets (GatedGCN) (Bresson & Laurent, 2017), etc.).

Despite such developments, all existing GNNs still suffer from two generic limitations: **(i)** they can only process graphs where each vertex is represented by a single value or a vector that has the same dimension as the other vertices; and **(ii)** they assume that each input graph is independent from others during the propagation process. However, an object represented by a vertex in the graph may not be well described by a single vector with fixed dimensionality, i.e., it may be better described by a graph or matrix (e.g., human face (Luo et al., 2022) and skeleton (Shi et al., 2019)). Previous studies frequently show that the relationship among graph samples provides informative cues for different graph analysis tasks. Such relationship cues can be represented by their correlations (Li et al., 2018), similarities (Zhong et al., 2021), and combination coefficients (Pan & Kang, 2021), etc. The main

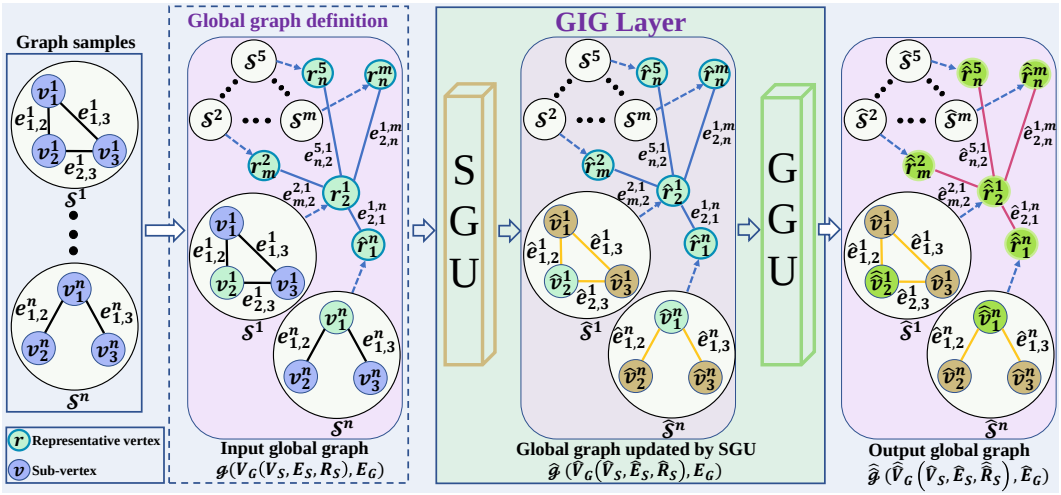


Figure 1: The pipeline of our GIG network and the GRM strategy. Given a set of input graph samples, we first combine them as a global graph, where each graph sample represented by a vertex/sub-graph and each pair of sub-graphs are connected by a global-edge (Sec. 3.2). We feed the global graph to the proposed GIG layer. Its sub-graph updating (SGU) module first individually updates each sub-graph, i.e., its sub-edges and sub-vertices (Sec. 3.1.1). Then, the global-graph updating (GGU) module further updates global-edges and representative vertices of each sub-graph to generate an updated global graph (Sec. 3.1.2).

shortcoming of these approaches is that they only extract the manually-defined relationships, which may ignore crucial task-specific cues that are not considered by their rules. In other words, it is worth investigating how to properly deep learn underlying task-specific relationship cues among target graph samples.

In this paper, we propose a novel Graph in Graph (GIG) Neural Network that aims to address the two limitations of existing GNNs discussed above, while retaining their advantages, i.e., the GIG can process heterogeneous graphs and graphs containing multi-dimensional edge features.**Specifically, the proposed GIG can process graphs, each of whose vertices is also represented by a graph (called sub-graph in this paper).** It consists of two main modules: sub-graph updating module (SGU) and global-graph updating (GGU) module. The SGU first individually updates edges and vertices of each sub-graph, extracting its internal task-related information. Then, the GGU exchanges messages among all sub-graphs by learning global-edges between each pair of sub-graphs. This way, the task-specific relationship cues between sub-graphs can be learned in an end-to-end manner. Building on our GIG, we further propose an GIG graph relationship modelling (GRM) strategy for general graph analysis tasks. It combines multiple input graph samples as a global graph, where each graph sample is treated as a vertex (sub-graph) in the global graph, and each pair of sub-graphs are connected by global-edges. Then, the GIG is utilized to make predictions from the obtained global graph. As a result, additional task-specific relationship cues among different graph samples would be modelled through global-edges during the GIG propagation process. The full pipeline of our GIG and GRM strategy are illustrated in Figure. 1. The main contributions and novelties of this paper are summarized as follows:

- We propose an Graph in Graph (GIG) Neural Network which can process graphs whose vertices can be represented by a set of heterogeneous graphs while each edge can be represented by a multi-dimensional vector. The vertex and edge feature updating mechanisms of the GIG are flexible to be customized by any existing vertex and edge updating algorithms. To the best of our knowledge, while existing GNNs can only process graphs whose vertices are single values or vectors, **the GIG is the first GNN model that can process graphs, each of whose vertices is also represented by a graph**.
- We propose an GIG graph relationship modelling (GRM) strategy that applies the GIG to model the task-specific relationship cues between multiple graph samples in an end-to-end manner, for generic graph analysis tasks.

- We show that the proposed GIG network and GIG relational learning strategy can be applied to various graph analysis tasks, where the standardised pre-processing and training strategy allows our approach to provide new state-of-the-art results on five out of seven evaluated graph benchmark datasets.

Different from GNNs designed for hypernode graph analysis (Yadati et al., 2019; Dong et al., 2020), whose edges can connect more than two vertices, the proposed GIG is desgined to process standard graphs, each of whose edges only connects a pair of vertices. The GIG also differs from the recently proposed subgraph GNNs (Alsentzer et al., 2020; Sun et al., 2021; Meng et al., 2018; Zhang et al., 2019b) which split a graph into multiple sub-graphs. Our GIG model does not split input graphs but directly processes input graphs, where these graph vertices can be represented by other graphs.

2 PRELIMINARIES

In this section, we formulate the general propagation mechanism of existing GNNs. Given a graph sample $\mathcal{G}(V, E)$, where V is the set of vertices and E is the set of edges, existing GNNs update as:

$$\hat{\mathcal{G}}(\hat{V}, \hat{E}) = \mathrm{f}(\mathcal{G}(V, E), A), \tag{1}$$

where A denotes the topology of the input graph sample. Let $\boldsymbol{v}_n \in V$ to denote a vertex and $\boldsymbol{e}_{n,m} = (\boldsymbol{v}_n, \boldsymbol{v}_m) \in E$ to denote an edge pointing from $\boldsymbol{v}_n$ to $\boldsymbol{v}_m$. The general function for updating edge feature $\boldsymbol{e}_{n,m}$ is formulated as:

$$\hat{\boldsymbol{e}}_{n,m} = \begin{cases} \mathrm{f}_e(\boldsymbol{e}_{n,m}, \boldsymbol{v}_n, \boldsymbol{v}_m) & \text{Updating edge} \\ \boldsymbol{e}_{n,m} & \text{Otherwise} \end{cases} \tag{2}$$

where f_e is a differentiable edge feature updating function fed with the initial edge feature $e_{n,m}$ and its connecting vertex features $\boldsymbol{v}_n$ and $\boldsymbol{v}_m$. Then, each vertex feature $\boldsymbol{v}_n$ is updated as:

$$\hat{\boldsymbol{v}}_n = \mathrm{f}_v(\boldsymbol{v}_n, \mathbf{m}_{\eta(\boldsymbol{v}_n)}), \text{ where } \mathbf{m}_{\eta(\boldsymbol{v}_n)} = \mathrm{Agg}(g_v(\boldsymbol{v}_m, \hat{\boldsymbol{e}}_{n,m}) | \boldsymbol{v}_m \in \eta(\boldsymbol{v}_n)). \tag{3}$$

where f_v is a differentiable vertex updating function inputted with $\boldsymbol{v}_n$ and its adjacent vertices $\eta(\boldsymbol{v}_n)$; Agg denotes an aggregation operation, where each adjacent vertex $\boldsymbol{v}_m$ impacts $\boldsymbol{v}_n$ through the corresponding edge $\boldsymbol{e}_{n,m}$ using the function g_v. In particular, the detailed formats of functions f_v, f_e g_v and Agg are individually specified by the employed GNN.

3 THE PROPOSED APPROACH

This section presents the theoretical details of our GIG network including vertex and edge updating strategies within and among sub-graphs as well as its differentiable analysis, in Sec. 3.1. Then, we propose an novel GIG graph relationship modelling (GRM) strategy that applies the GIG to extract task-specific relationship cues among graph samples (Sec. 3.2) for generic graph analysis tasks.

Specifically, we define a new graph structure, called "Graph in Graph", as $\mathcal{G}(V_G, E_G)$. Unlike conventional graph structures, each of whose vertices is represented by a single value or a vector, each vertex in $\mathcal{G}(V_G, E_G)$ is represented by a graph $\mathcal{S}^i(V_S^i, E_S^i)$ (i.e., $V_G = \{\mathcal{S}^1(V_S^1, E_S^1), \mathcal{S}^2(V_S^2, E_S^2), \cdots, \mathcal{S}^I(V_S^I, E_S^I)\}$). In this paper, we call $\mathcal{G}(V_G, E_G)$ the ***global graph*** and each $\mathcal{S}^i(V_S^i, E_S^i)$ a ***sub-graph***. Here, we define the sub-graph $\mathcal{S}^i$ has N_i sub-vertices $V_S^i = \{\boldsymbol{v}_1^i, \boldsymbol{v}_2^i, \cdots, \boldsymbol{v}_{N_i}^i\}$, where each sub-vertex feature is a $1 \times D$ dimensional vector and each sub-edge $\boldsymbol{e}_{n,m}^i \in E_S^i$ connects a pair of sub-vertices in $\mathcal{S}^i$. Meanwhile, each global-edge $\boldsymbol{e}_{n,m}^{i,j}$ is defined to connect a pair of sub-graphs $\mathcal{S}^i$ and $\mathcal{S}^j$. In particular, each sub-graph $\mathcal{S}^i$ is represented by one or multiple representative sub-vertices $\boldsymbol{r}_n^i = \boldsymbol{v}_n^i \in V_S^i$, and each global-edge $\boldsymbol{e}_{n,m}^{i,j}$ connects a pair of representative sub-vertices $\boldsymbol{r}_n^i$ and $\boldsymbol{r}_m^j = \boldsymbol{v}_m^j \in V_S^j$. As a result, the input graph can be also defined as $\mathcal{G}(V_G(V_S, E_S, R_S), E_G)$, where $V_S = \{V_S^1, V_S^2, \cdots, V_S^I\}$, $E_S = \{E_S^1, E_S^2, \cdots, E_S^I\}$ and R_S is a subset of V_S (i.e., $R_S^i \in V_S^i$). We additionally provide the GIG pseudocode in the appendix A.3.

3.1 THE GIG LAYER

Given a global graph $\mathcal{G}(V_G(V_S, E_S, R_S), E_G)$, the GIG conducts two-stage edge and vertex updating. The first stage (i.e., **sub-graph level updating (SGU)**, Figure 2) only updates each sub-graph

individually, which can be formulated as follows:

$$\hat{\mathcal{G}}(\hat{V}_G(\hat{V}_S, \hat{E}_S, \hat{R}_S), E_G) = \text{SGU}(\mathcal{G}(V_G(V_S, E_S, R_S), E_G)) \tag{4}$$

where $\hat{V}_S$ and $\hat{E}_S$ denote the updated sub-vertex features and sub-edge features (i.e., since $R_S \in V_S$, they will also be updated within each sub-graph). Then, the second stage (**global-graph level updating (GGU)**, Figure 3) not only updates global-edges but also further updates representative sub-vertices as:

$$\hat{\hat{\mathcal{G}}}(\hat{\hat{V}}_G(\hat{V}_S, \hat{E}_S, \hat{\hat{R}}_S), \hat{E}_G) = \text{GGU}(\hat{\mathcal{G}}(\hat{V}_G(\hat{V}_S, \hat{E}_S, \hat{R}_S), E_G)) \tag{5}$$

where $\hat{E}_G$ is the updated global edge features; $\hat{\hat{R}}_S$ denotes representative sub-vertices which are connected by global-edges, and thus have been further updated in the GGU.

3.1.1 SUB-GRAPH LEVEL VERTEX AND EDGE UPDATING

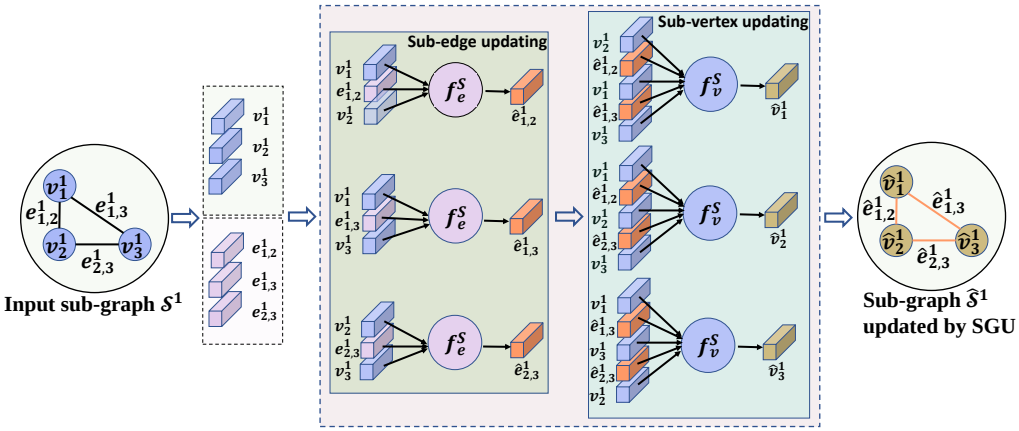


Figure 2: Illustration of the SGU module, which updates each sub-graph individually. For each sub-graph, the SGU first updates all of its sub-edges, and then updates all sub-vertices.

Let the input to the SGU module be a global graph $\mathcal{G}\{\mathcal{S}^1, \mathcal{S}^2, \cdots, \mathcal{S}^I, E_G\}$, where I is the number of vertices (sub-graphs) contained in $\mathcal{G}$. The SGU individually updates each sub-graph $\mathcal{S}^i$, including its sub-vertex and sub-edge features. Specifically, it updates the sub-edge features $\boldsymbol{e}^i_{n,m}$ as the $\hat{\boldsymbol{e}}^i_{n,m}$ by considering: (i) the input sub-edge feature $\boldsymbol{e}^i_{n,m}$; and (ii) the corresponding sub-vertex features $\boldsymbol{v}^i_n$ and $\boldsymbol{v}^i_m$, which can be formulated as:

$$\hat{\boldsymbol{e}}^i_{n,m} = f^S_e(\boldsymbol{e}^i_{n,m}, \boldsymbol{v}^i_n, \boldsymbol{v}^i_m) \tag{6}$$

where f^S_e represents the edge feature updating strategy, which can be customized based on the edge updating algorithm used in any existing GNNs (e.g., GatedGCN and GAT).

Then, the GIG layer updates each sub-vertex feature (including representative sub-vertices) $\boldsymbol{v}^i_n$ as the $\hat{\boldsymbol{v}}^i_n$ by considering: (i) the input sub-vertex feature $\boldsymbol{v}^i_n$; (ii) $\boldsymbol{v}^i_n$'s adjacent sub-vertex features $\boldsymbol{v}^i_m \in \mathcal{N}_{\text{sub}}(\boldsymbol{v}^i_n)$ in the corresponding sub-graph $\mathcal{G}^i$; and (iii) all updated sub-edge features $\hat{\boldsymbol{e}}^i_{n,m}$ that connect the $\boldsymbol{v}^i_n$ and its adjacent sub-vertices. This process can be formulated as:

$$\hat{\boldsymbol{v}}^i_n = f^S_v(\boldsymbol{v}^i_n, \text{Agg}(g_v(\hat{\boldsymbol{e}}^i_{n,m}, \boldsymbol{v}^i_m) | \boldsymbol{v}_m \in \mathcal{N}_{\text{sub}}(\boldsymbol{v}^i_n)) \tag{7}$$

where f^S_v is the differentiable vertex feature updating function; g_v is a differentiable message passing function that passes the impact of each vertex feature to its neighbours; and Agg denotes the aggregation operation. These functions can also be customized from any existing GNNs.

3.1.2 GLOBAL-GRAPH LEVEL VERTEX AND EDGE UPDATING

After updating each sub-graph individually, the GIG then performs global-level updating for all sub-graphs and global-edges, which models the global-level relationships among all sub-graphs. Given

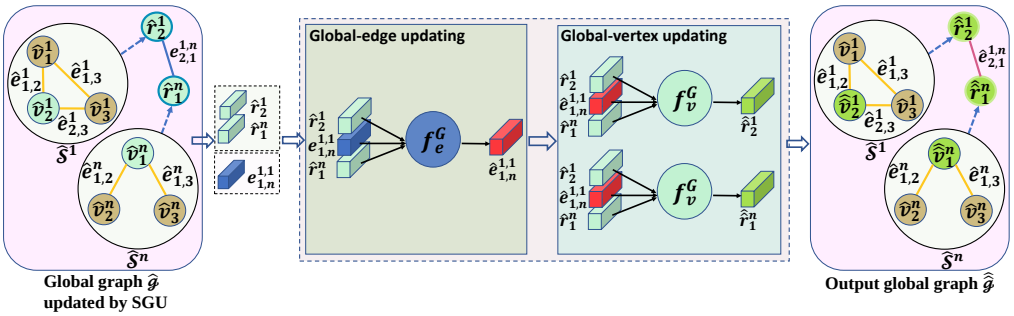


Figure 3: The GGU module first updates each global-edge, and then updates each sub-graphs (i.e., updating related sub-vertices in both sub-graphs, which are connected by global-edges).

a global-edge $\boldsymbol{e}^{i,j}_{n,m} \in E_G$ that connects a pair of sub-graphs $\hat{\mathcal{S}}^i$ and $\hat{\mathcal{S}}^j$ that have been updated by SGU, the GGU first updates it as the $\hat{\boldsymbol{e}}^{i,j}_{n,m}$ using a differentiable global-edge updating function f^G_e, which can be formulated as:

$$\hat{\boldsymbol{e}}^{i,j}_{n,m} = f^G_e(\boldsymbol{e}^{i,j}_{n,m}, \hat{\boldsymbol{r}}^i_n, \hat{\boldsymbol{r}}^j_m). \tag{8}$$

where $\hat{\boldsymbol{r}}^i_n \in \mathcal{S}^i$ and $\hat{\boldsymbol{r}}^j_m \in \mathcal{S}^j$ $(i \neq j)$ are corresponding representative sub-vertices updated in SGU which are connected by the global-edge $\boldsymbol{e}^{i,j}_{n,m}$.

After updating all global-edges, the GGU then updates sub-graphs by updating each representative sub-vertex feature $\hat{\boldsymbol{v}}^i_n$ as the $\hat{\hat{\boldsymbol{v}}}^i_n$, using a differentiable global vertex updating function f^G_v based on: (i) the representative sub-vertex feature $\hat{\boldsymbol{v}}^i_n$ that has been updated at the sub-graph level; (ii) the corresponding representative sub-vertex features of its global-level neighbours $\mathcal{N}_{\text{global}}(\hat{\boldsymbol{r}}^i_n)$; and (iii) the corresponding updated global-edge features $\hat{\boldsymbol{e}}^{i,j}_{n,m}$ which connect $\hat{\boldsymbol{r}}^i_n$ with its global-level neighbour sub-vertices. This procedure can be represented as:

$$\hat{\hat{\boldsymbol{r}}}^i_n = f^G_v(\hat{\boldsymbol{r}}^i_n, \text{Agg}(g_v(\hat{\boldsymbol{e}}^{i,j}_{n,m}, \hat{\boldsymbol{r}}^j_m) | \hat{\boldsymbol{r}}^j_m \in \mathcal{N}_{\text{global}}(\hat{\boldsymbol{r}}^i_n)) \tag{9}$$

Since all edge and vertex updating functions in the proposed GIG are flexible, we leverage the edge and vertex updating strategies of widely-sued GAT and GatedGCN provided in (Dwivedi et al., 2020) in this paper, which allow the GIG to process heterogeneous graphs and graphs whose edges are represented by multi-dimensional vectors.

In summary, the SGU and GGU iteratively update sub-graphs and the global graph, which facilitate the messages exchanged among isolated sub-graphs, allowing their relationship cues to be utilized for the analysis.

3.1.3 Differentiation analysis of the GIG layer

In this section, we provide the back-propagation analysis of the proposed GIG layer. Supposing that the loss value $\mathcal{L}$ is computed by the objective function ℓ; the graph $\hat{\tilde{\mathcal{G}}}(\hat{\tilde{V}}_G(\hat{\tilde{V}}_S, \hat{E}_S, \hat{\tilde{R}}_S), \hat{E}_G)$ is the output of the GIG layer, the gradient of the proposed GIG w.r.t. the input graph $\mathcal{G}(V_G(V_S, E_S, R_S), E_G)$ $(R_S \in E_S)$ can be defined as:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{G}(V_G(V_S, E_S, R_S), E_G)} = \frac{\partial \mathcal{L}}{\partial V_S} + \frac{\partial \mathcal{L}}{\partial E_S} + \frac{\partial \mathcal{L}}{\partial R_S} + \frac{\partial \mathcal{L}}{\partial E_G} \tag{10}$$

Specifically, the $\frac{\partial \mathcal{L}}{\partial E_S}$, $\frac{\partial \mathcal{L}}{\partial V_S}$ and $\frac{\partial \mathcal{L}}{\partial E_G}$ can be computed as:

$$\frac{\partial \mathcal{L}}{\partial E_S} = \frac{\partial \mathcal{L}}{\partial \hat{V}_S}\frac{\partial \hat{V}_S}{\partial \hat{E}_S}\frac{\partial \hat{E}_S}{\partial E_S} + \frac{\partial \mathcal{L}}{\partial \hat{\tilde{R}}_S}\frac{\partial \hat{\tilde{R}}_S}{\partial \hat{R}_S}\frac{\partial \hat{R}_S}{\partial \hat{E}_S}\frac{\partial \hat{E}_S}{\partial E_S} + \frac{\partial \mathcal{L}}{\partial \hat{\tilde{R}}_S}\frac{\partial \hat{\tilde{R}}_S}{\partial \hat{E}_G}\frac{\partial \hat{E}_G}{\partial \hat{R}_S}\frac{\partial \hat{R}_S}{\partial \hat{E}_S}\frac{\partial \hat{E}_S}{\partial E_S} \tag{11}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial V_S} = &\frac{\partial \mathcal{L}}{\partial \hat{V}_S}\frac{\partial \hat{V}_S}{\partial V_S} + \frac{\partial \mathcal{L}}{\partial \hat{V}_S}\frac{\partial \hat{V}_S}{\partial \hat{E}_S}\frac{\partial \hat{E}_S}{\partial V_S} + \frac{\partial \mathcal{L}}{\partial \hat{\tilde{R}}_S}\frac{\partial \hat{\tilde{R}}_S}{\partial \hat{R}_S}\frac{\partial \hat{R}_S}{\partial V_S} + \frac{\partial \mathcal{L}}{\partial \hat{\tilde{R}}_S}\frac{\partial \hat{\tilde{R}}_S}{\partial \hat{R}_S}\frac{\partial \hat{R}_S}{\partial \hat{E}_S}\frac{\partial \hat{E}_S}{\partial V_S} \\ &+ \frac{\partial \mathcal{L}}{\partial \hat{\tilde{R}}_S}\frac{\partial \hat{\tilde{R}}_S}{\partial \hat{E}_G}\frac{\partial \hat{E}_G}{\partial \hat{R}_S}\frac{\hat{R}_S}{V_S} + \frac{\partial \mathcal{L}}{\partial \hat{\tilde{R}}_S}\frac{\partial \hat{\tilde{R}}_S}{\partial \hat{E}_G}\frac{\partial \hat{E}_G}{\partial \hat{R}_S}\frac{\partial \hat{R}_S}{\partial \hat{E}_S}\frac{\partial \hat{E}_S}{\partial V_S}\end{aligned} \tag{12}$$

$$\frac{\partial \mathcal{L}}{\partial E_G} = \frac{\partial \mathcal{L}}{\partial \hat{E}_G}\frac{\partial \hat{E}_G}{\partial E_G} + \frac{\partial \mathcal{L}}{\partial \hat{E}_G}\frac{\partial \hat{E}_G}{\partial \hat{R}_S}\frac{\partial \hat{R}_S}{\partial V_S} \tag{13}$$

where $\hat{V}_S$, $\hat{R}_S \in \hat{V}_S$, $\hat{\hat{R}}_S$, $\hat{E}_S$ and $\hat{E}_G$ are produced by differentiable vertex and edge updating functions (f_e^S, f_v^S, f_e^G and f_v^G) from pre-defined inputs V_S, E_S and E_G, while $\mathcal{L}$ is produced by the differentiable loss function ℓ whose input is $\hat{\hat{\mathcal{G}}}$. As a result, the proposed GIG layer is a fully differentiable GNN layer, and can be stacked as a deep GNN model.

3.2 GIG GRAPH RELATIONSHIP MODELLING FOR GENERIC GRAPH ANALYSIS TASKS

In this section, we show how to apply the proposed GIG to general graph analysis tasks. The GRM strategy aims at extracting additional task-specific relationship cues among a batch of input graph samples to improve graph analysis performances.

Global graph definition: Given a set of graph samples $\mathcal{S}^1, \mathcal{S}^2, \cdots, \mathcal{S}^I$, we first combine them as a global graph $\mathcal{G}(V_G, E_G)$ that has I vertices, where each vertex (sub-graph) V_G^i represents a graph sample $\mathcal{S}^i(V_S^i, E_S^i)$. Since our GIG can process heterogeneous graphs, these graph samples can have different typologies. We then define the topology of the global graph by: (i) keeping the original internal typologies for each graph sample and (ii) additionally adding one/multiple global-edges to connect each pair of sub-graphs. In particular, for each pair of graph samples $\mathcal{S}^i$ and $\mathcal{S}^j$, we first define one/multiple representative vertices $\boldsymbol{r}_{n'}^i \in V_S^i$ and $\boldsymbol{r}_{m'}^j \in V_S^j$ as their representations. Then, we define each global-edge $\boldsymbol{e}_{n',m'}^{i,j}$ by connecting a pair of representative sub-vertices, whose initial feature is defined as their dot product (i.e., $\boldsymbol{e}_{n',m'}^{i,j} = \langle \boldsymbol{r}_{n'}^i, \boldsymbol{r}_{m'}^j \rangle$). The experiments (Sec. 4) show that our GIG is robust to various representative vertex and global-edge definition strategies.

GIG-based graph relationship modelling and analysis: We then feed the produced global graph to the GIG to model the relationship between graph samples. Subsequently, the sub-graph $\hat{\mathcal{S}}^i$ produced from each graph sample $\mathcal{S}^i$ contains task-specific cues extracted from not only the $\mathcal{S}^i$ itself but also its relationship with other graph samples. Specifically, each sub-edge feature $\boldsymbol{e}_{n,m}^i$ contained in the graph sample $\mathcal{S}^i$ as well as each global-edge feature $\boldsymbol{r}_{n',m'}^{i,j}$ connecting graph samples $\mathcal{S}^i$ and $\mathcal{S}^j$ in the global graph would be updated as:

$$\hat{\boldsymbol{e}}_{n,m}^i = f_e^S(\boldsymbol{e}_{n,m}^i, \boldsymbol{v}_n^i, \boldsymbol{v}_m^i) \quad \& \quad \hat{\boldsymbol{e}}_{n',m'}^{i,j} = f_e^G(\boldsymbol{e}_{n',m'}^{i,j}, \boldsymbol{r}_{n'}^i, \boldsymbol{r}_{m'}^j) \tag{14}$$

In comparison to the f_e of Eqa. 2, which represents the general edge updating mechanism of existing GNNs, a set of global-edges are additionally learned by our f_e^G in order to exchange messages among different graph samples. In addition to updating vertices contained in each graph sample using f_v^S, the GIG further updates each representative vertex v_n^i as:

$$\begin{aligned}
&\hat{\boldsymbol{r}}_n^i = f_v^G(\boldsymbol{r}_n^i, \mathbf{m}_{\eta_{\text{sub}}(\boldsymbol{r}_n^i)}, \mathbf{m}_{\eta_{\text{global}}(\boldsymbol{r}_n^i)}), \ \text{ where} \\
&\mathbf{m}_{\eta_{\text{sub}}(\boldsymbol{r}_n^i)} = \text{Agg}(g_v(\boldsymbol{r}_n^i, \hat{\boldsymbol{e}}_{n,m}^i) | \boldsymbol{r}_m^i \in \eta_{\text{sub}}(\boldsymbol{r}_n^i)); \\
&\mathbf{m}_{\eta_{\text{global}}(\boldsymbol{r}_n^i)} = \text{Agg}(g_v(\boldsymbol{r}_{m'}^j, \hat{\boldsymbol{e}}_{n,m'}^{i,j}) | \boldsymbol{r}_{m'}^j \in \eta_{\text{global}}(\boldsymbol{r}_n^i))
\end{aligned} \tag{15}$$

In comparison to the vertex feature extracted by existing GNNs (Eqa. 3), our GIG updates each representative vertex feature by additionally considering its relationship with global-level neighbours that belong to other graph samples. In other words, as an end-to-end model, the output generated by our GIG contains additional task-specific cues extracted from $\mathbf{m}_{\eta_{\text{global}}(\boldsymbol{r}_n^i)}$. Thus, it would theoretically improve the graph analysis performances achieved by existing GNNs that only consider the internal cues contained in each graph sample.

4 EXPERIMENT

Datasets: We evaluate the proposed approach on a set of widely-used medium-size graph analysis benchmark datasets (Dwivedi et al., 2020), which are: (i) MNIST (LeCun et al., 1998) and CIFAR10 (Krizhevsky et al., 2009) datasets for graph classification task; (ii) ZINC (Irwin et al., 2012) and AQSOL (Sorkun et al., 2019) datasets for graph regression task; (iii) PATTERN (Dwivedi et al., 2020) and CLUSTER (Dwivedi et al., 2020) datasets for node classification task; and (iv) TSP

(Joshi et al., 2022) dataset for link prediction task. Since our proposed GIG targets at improving performance by promoting information messaging among independent single graphs, we did not evaluate on OGBL-COLLAB (Hu et al., 2020) and WikiCS (Mernyei & Cangea, 2007) with only one single graph, and Cycles (Loukas, 2019) and CSL (Murphy et al., 2019) for evaluating Graph Positional Encoding.

Implementation details: To facilitate fair comparative evaluation, we follow the same data split, optimizer and loss functions suggested by (Dwivedi et al., 2020) for all experiments. To show the generalization ability of the GIG, we instantiate it based on vertex and edge updating mechanisms of two widely-used GNN models implemented in (Dwivedi et al., 2020): GatedGCN (the default setting) and GAT, where SGU and GGU modules share the same vertex and edge updating functions. Specially, we build a batch of input graphs as a global graph by setting each graph sample as a sub-graph. We additionally provide: (i) detailed description of datasets; (ii) more training details; and (iii) runtime analysis of the GIG, in the appendix.

Metrics: We follow the same metrics settings as Dwivedi et al. (2020) to evaluate: (i) the graph-level classification accuracy on MNIST and CIFAR10; (ii) the Mean Absolute Error (MAE) on ZINC and AQSOL; (iii) the average vertex-level classification accuracy on PATTERN and CLUSTER; and (iv) the F1 score of the binary link classification on TSP.

4.1 Comparative Results

Following (Dwivedi et al., 2020), we evaluate the performance of our GIG on four types of graph analysis tasks: graph classification and regression, vertex classification, and link prediction, which cover all typical graph analysis tasks (graph-level, vertex-level and edge-level analysis).

Table 1: Experimental results on 7 benchmark datasets from (Dwivedi et al., 2020). Highlighted are the top 1st, 2nd and 3rd results.

Task	Graph classification		Graph regression		Node classification		Edge classification
Dataset / Model	MNIST	CIFAR10	ZINC	AQSOL	PATTERN	CLUSTER	TSP
	Test Acc(%) ↑		Test MAE ↓		Test Acc(%) ↑		Test F1↑
GCN	90.12 ± 0.15	54.14 ± 0.39	0.278 ± 0.003	$\mathbf{1.333 \pm 0.013}$	85.61 ± 0.03	69.03 ± 1.37	0.643 ± 0.001
GIN	96.49 ± 0.25	55.26 ± 1.53	0.387 ± 0.015	1.894 ± 0.024	85.59 ± 0.01	64.72 ± 1.55	0.656 ± 0.003
GAT	95.54 ± 0.21	64.22 ± 0.46	0.384 ± 0.007	1.403 ± 0.008	78.27 ± 0.19	70.59 ± 0.45	0.671 ± 0.002
GatedGCN	97.34 ± 0.14	67.31 ± 0.31	0.214 ± 0.013	$\mathbf{0.996 \pm 0.008}$	86.51 ± 0.09	76.08 ± 0.20	0.838 ± 0.002
PNA	97.94 ± 0.12	70.86 ± 0.27	0.188 ± 0.004	-	86.57 ± 0.08	-	-
DGN	-	$\mathbf{72.84 \pm 0.42}$	0.168 ± 0.003	-	$\mathbf{86.68 \pm 0.03}$	-	-
EGT	$\mathbf{98.17 \pm 0.09}$	68.70 ± 0.41	$\mathbf{0.108 \pm 0.009}$	-	$\mathbf{86.82 \pm 0.02}$	$\mathbf{79.23 \pm 0.35}$	$\mathbf{0.853 \pm 0.001}$
ARGNP	-	$\mathbf{73.90 \pm 0.15}$	$\mathbf{0.136 \pm 0.002}$	-	-	$\mathbf{77.35 \pm 0.05}$	$\mathbf{0.855 \pm 0.001}$
GNAS-MP	$\mathbf{98.01 \pm 0.10}$	70.10 ± 0.44	0.242	-	$\mathbf{86.80 \pm 0.10}$	62.21 ± 0.20	-
GIG(Ours)	$\mathbf{98.72 \pm 0.01}$ $\mathbf{0.55 \uparrow}$	$\mathbf{75.66 \pm 0.06}$ $\mathbf{1.76 \uparrow}$	$\mathbf{0.157 \pm 0.001}$	$\mathbf{0.975 \pm 0.011}$ $\mathbf{0.21 \downarrow}$	$\mathbf{86.82 \pm 0.01}$	$\mathbf{78.95 \pm 0.07}$	$\mathbf{0.860 \pm 0.001}$ $\mathbf{0.05 \uparrow}$

Competitors: We first compare our GIG with widely-used/recently proposed GNNs on seven benchmark graph datasets in Table 1, including (i) GCN (Kipf & Welling, 2016) and GIN(Xu et al., 2018) that only consider binary adjacency relationship among internal sub-vertices in each graph sample without using multi-dimensional edge features; (ii) GAT(Veličković et al., 2017), GatedGCN (Bresson & Laurent, 2017), PNA(Corso et al., 2020), DGN (Beaini et al., 2021), EGT (Hussain et al., 2022) and ARGNP(Cai et al., 2022) that can process multi-dimensional edge features contained in each graph samples; and (iii) GNAS-MP (Cai et al., 2021) that applies Neural Architecture Search (NAS) to explore an optimal GNN architecture for each dataset.

Results: According to comparison results, we make the following observations: **(i)** GIG achieved clear performance gains in 4/7 datasets, and superior performances in the rest. This suggests that the advantage of GIG is scalable to different tasks. Although GIG ranks the 3rd in ZINC, it still improves its baseline GatedGCN from 0.214 to 0.157. A plausible reason of not outperforming EGT and ARGNP could be the operation that averages all single-value vertex features on ZINC adopted by (Dwivedi et al., 2020), which limits capability to carry the learned relationship cues among graph samples. **(ii)** Comparing GIG with its baseline GatedGCN, we found that the GIG obtained clear improvements on all datasets, with absolute 3.3% average improvements. This directly confirms the effectiveness of sub-graph communication in the proposed GIG, which also maintains superiority of GatedGCN that uses multi-dimensional edge features. **(iii)** GIG outperforms GNAS-MP which applies NAS to explore task-specific weights and optimal architecture. This indicates great potential

of proposed GIG to push the state-of-the-art performance on this benchmark further with other more advanced architectures or vertex/edge updating strategies.

4.2 Ablation Studies and Good Properties

We further conduct a series of ablation studies on all graph-level, vertex-level and edge-level analysis tasks (i.e., one dataset is employed for each task) to investigate the influences of two main variables of our approach: (i) the number of employed GIG layers; and (ii) the number of sub-graphs/graph samples that are used to construct the global graph in our GRM strategy. Moreover, we also show that our approach is not only robust to various representative vertex and global-edge definition strategies but also flexible to be customized by different edge/vertex updating algorithms while achieving clear enhanced performances.

The number of graph samples contained in a global-graph: We demonstrate the influence of the number of graph samples used to construct the global graph in Figure 4. It can be observed that the optimal number is depending on the employed task and dataset. However, the performances achieved by GIG are relatively robust to the number of sub-graphs contained in the input global graphs. This validates that GIG can effectively extract task-specific relationship cues among multiple input graph samples as well as extracting their internal task-specific cues, regardless of their number.

Figure 4: Results achieved from global graphs that have different numbers of vertices (sub-graphs/graph samples), where each bracket denotes (the number of vertices in each global graph, the prediction result).

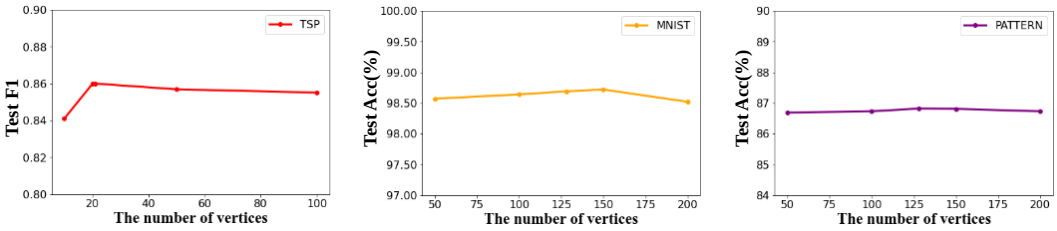


Table 2: Experimental results achieved for different numbers of stacked GIG layers.

Dataset Number of layers	MNIST Test Acc(%) ↑	PATTERN	TSP Test F_1 ↑
1	**98.72 ± 0.01**	86.66 ± 0.01	0.839 ± 0.000
2	98.45 ± 0.03	**86.82 ± 0.01**	0.851 ± 0.000
3	98.52 ± 0.02	86.75 ± 0.01	**0.860 ± 0.001**

Table 3: Experimental results achieved for different edge and vertex updating algorithms.

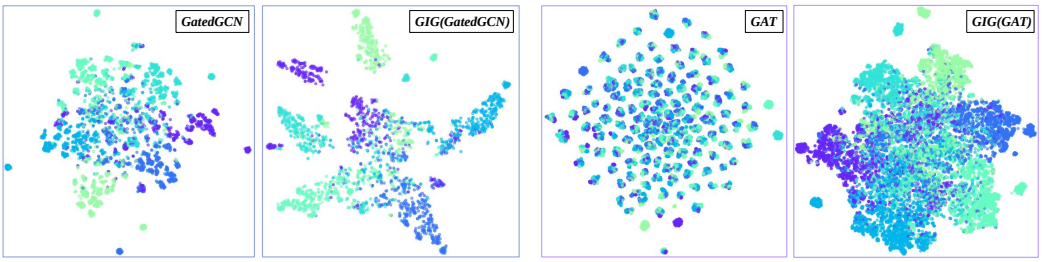
Dataset Model	MNIST Test Acc(%) ↑	PATTERN	TSP Test F_1 ↑
GAT	95.54 ± 0.45	78.27 ± 0.19	0.671 ± 0.002
GIG(GAT)	**97.91 ± 0.02**	**78.98 ± 0.01**	**0.819 ± 0.001**
GatedGCN	97.34 ± 0.20	86.51 ± 0.09	0.838 ± 0.002
GIG(GatedGCN)	**98.72 ± 0.01**	**86.82 ± 0.01**	**0.860 ± 0.001**

The number of employed GIG layers (Model depth): It can be observed from Table 2 that although the best settings for different tasks are not the same (i.e., the optimal number of GIG layers should be employed for MNIST, PATTERN and TSP datasets are one, two and three, respectively), the graph analysis performances did not changed dramatically when changing the number of the stacked GIG layers. In particular, when stacking three different numbers (1, 2 and 3) of GIG layers, the results achieved for graph classification and vertex classification tasks are only varied less than 0.3%, indicating that our GIG model is relatively robust to the number of the stacked GIG layers.

Flexible to be customized by different edge and vertex updating algorithms: As displayed in Table 3, our GIG's edge and vertex features' updating mechanisms can be customized by different algorithms used in existing GNNs (e.g., GatedGCN and GAT). More importantly, based on the same edge and vertex mechanisms, the GIG show clear advantages over the original GatedGCN and GAT networks on all three types of graph analysis tasks, providing average 1.47% and 8.48% improvements, respectively. Figure 5 compares the latent representations extracted by GIG which employed different vertex/edge updating strategies, where GIG jointly processes a batch of graph samples by combine them as a global graph and GatedGCN/GAT individually processes them without considering their relationships. It is clear that the latent representations learned by GIG are more discrminative. These results again suggest that the proposed GIG can additionally encode task-

specific cues from the relationships between sub-graphs to enhance graph analysis performances, when using different edge/vertex updating algorithms.

Figure 5: TSNE visualisation of features learned on CLUSTER dataset (a six classes vertex classification dataset), where each dot represents a vertex and each colour represents a category label. Each pair of GNNs share the same vertex/edge updating functions.



Robust to different representative vertex and global-edge definition settings: We evaluate four types of representative vertex and global-edge definition strategies: (i) each vertex in a graph sample is connected with the top-K most similar vertices in all graph samples; (ii) for each pair of graph samples $\mathcal{S}^i$ and $\mathcal{S}^j$, each vertex $\boldsymbol{v}_n^i \in \mathcal{S}^i$ connects to its most similar vertex in $\mathcal{S}^j$; (iii) only using a pair of most similar vertices in two graph samples to define one global-edge between each pair of graph samples; (iv) randomly selecting a pair of vertex in two graph samples to define one global-edge between them. Consequently, all vertices in a graph sample are selected as the representative vertices and a pair of graph samples are connected by multiple global-edges in setting (i) and (ii), while only one vertex in each graph sample is selected as the representative vertex and a pair of graph samples are connected by one global-edge in setting (iii) and (iv). As displayed in Table 4 and Table 5, all settings produced similar results which are clearly better than the baseline that only processes each graph sample individually. Particularly, using one representative vertex to describe each graph sample not only reduces the complexity (i.e., the number of global-edges) of the global graph, but also achieved the best results on all datasets. This suggests that the underlying relationship among graph samples contain crucial task-specific cues for different graph analysis tasks, which can be effectively extracted by the proposed GRM strategy and GIG network in an end-to-end manner. More importantly, the GIG is robust to various representative vertex generation and global-edge definition strategies, which further indicates that the SGU and GGU can iteratively pass the crucial messages to sub-vertices of all sub-graphs regardless of the employed representative vertex generation and global connection strategies.

Table 4: Experimental results achieved for various representative vertex and global-edge definition settings.

Dataset Setting	MNIST Test Acc(%) ↑	PATTERN Test Acc(%) ↑	TSP Test F_1 ↑
Baseline	97.34 ± 0.20	86.51 ± 0.09	0.838 ± 0.002
(i)	98.47 ± 0.02	86.19 ± 0.01	-
(ii)	98.71 ± 0.03	86.70 ± 0.01	0.846 ± 0.001
(iii)	$\mathbf{98.72 \pm 0.01}$	$\mathbf{86.82 \pm 0.01}$	$\mathbf{0.860 \pm 0.001}$
(iv)	98.63 ± 0.02	86.79 ± 0.01	0.856 ± 0.000

Table 5: Average number of global-edges in each global graph for various representative vertex and global-edge definition settings.

Dataset Setting	MNIST 150 sub-graphs	PATTERN 128 sub-graphs	TSP 27 sub-graphs
Baseline	0	0	0
(i)	~ 6757693	~ 19330648	~ 3296097
(ii)	~ 1585276	~ 1924012	~ 200318
(iii)	~ 22466	~ 16180	~ 727
(iv)	~ 22466	~ 16180	~ 727

5 Conclusion

This paper proposes the first GNN model that can process graphs whose vertices are also represented by graphs, and additionally applies it to enhance general graph analysis performances by exploring task-specific relationship cues among input graph samples. The results show that our approach not only provided new state-of-art-results for various graph analysis tasks, but it is also robust to different global graph sizes (sub-graph numbers), model depths and global-edge/representative vertex definition strategies. Additionally, its vertex/edge updating function is customizable from existing GNNs. One limitation of this work is that we did not evaluate our GIG for multiple object data-based analysis, where each object in a data sample can be further represented as a graph rather than a vector. This will be addressed by our future work.

REPRODUCIBILITY STATEMENT

To allow the proposed GIG and related experiments to be easily reproduced or extended by future researchers, we provide: (i) all models' settings; (ii) all training hyper-parameter settings and training loss curves; and (iii) the software and hardware details employed in our experiments in the appendix, where the coding platform and dependencies are coming from standard deep learning and graph analysis python libraries. In addition, we provide our training, validation, testing codes and the weights of our best models in the supplementary material, with a detailed README file to guide users to run experiments and re-produce the results.

REFERENCES

Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *NeurIPS*, 2020.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *NeurIPS*, 2016.

Dominique Beaini, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. In *ICML*, 2021.

Xavier Bresson and Thomas Laurent. Residual Gated Graph ConvNets. *arXiv e-prints*, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Shaofei Cai, Liang Li, Jincan Deng, Beichen Zhang, Zheng-Jun Zha, Li Su, and Qingming Huang. Rethinking Graph Neural Architecture Search from Message-passing. *arXiv e-prints*, 2021.

Shaofei Cai, Liang Li, Xinzhe Han, Jiebo Luo, Zheng-Jun Zha, and Qingming Huang. Automatic relation-aware graph network proliferation. In *CVPR*, 2022.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal Neighbourhood Aggregation for Graph Nets. *arXiv e-prints*, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *NeurIPS*, 2016.

Yihe Dong, Will Sawin, and Yoshua Bengio. Hnhn: hypergraph networks with hyperedge neurons. *arXiv preprint arXiv:2006.12278*, 2020.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *CVPR*, 2019.

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. *NeurIPS*, 2020.

Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 2012.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 2022.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv e-prints*, 2016.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *AAAI*, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

Cheng Luo, Siyang Song, Weicheng Xie, Linlin Shen, and Hatice Gunes. Learning multi-dimensional edge feature-based au relation graph for facial action unit recognition. In *IJCAI*, 2022.

Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*, 2017.

Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. Subgraph pattern neural networks for high-order graph evolution prediction. In *AAAI*, 2018.

P Mernyei and C Wiki-CS Cangea. A wikipedia-based benchmark for graph neural networks. arxiv 2020. *arXiv preprint arXiv:2007.02901*, 2007.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *ICML*, 2019.

Erlin Pan and Zhao Kang. Multi-view contrastive graph clustering. *NeurIPS*, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *TPAMI*, 1990.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 2008.

Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *CVPR*, 2019.

Murat Cihan Sorkun, Abhishek Khetan, and Süleyman Er. Aqsoldb, a curated reference set of aqueous solubility and 2d descriptors for a diverse set of compounds. *Scientific data*, 2019.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *NeurIPS*, 2016.

Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Philip S Yu, and Lifang He. Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In *Proceedings of the Web Conference 2021*, 2021.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *arXiv e-prints*, 2017.

Kunze Wang, Soyeon Caren Han, Siqu Long, and Josiah Poon. Me-gcn: Multi-dimensional edge-embedded graph convolutional networks for semi-supervised text classification. *arXiv preprint arXiv:2204.04618*, 2022.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019a.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, 2019b.

Chao Xiong, Wen Li, Yun Liu, and Minghui Wang. Multi-dimensional edge features graph neural network on few-shot image classification. *IEEE Signal Processing Letters*, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? *arXiv e-prints*, 2018.

Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *NeurIPS*, 2019.

Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019a.

Zhihong Zhang, Dongdong Chen, Jianjia Wang, Lu Bai, and Edwin R Hancock. Quantum-based subgraph convolutional neural networks. *PR*, 2019b.

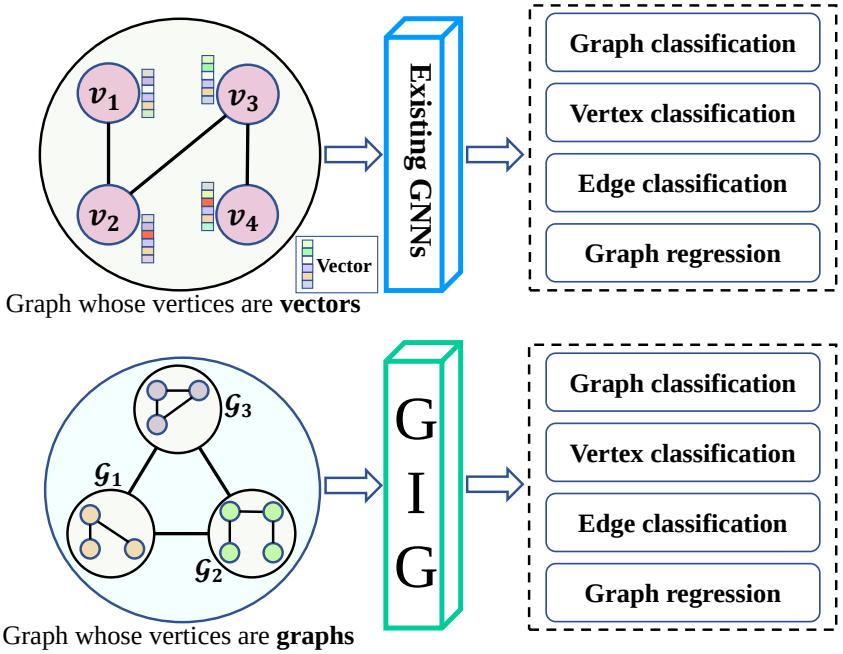
Huasong Zhong, Jianlong Wu, Chong Chen, Jianqiang Huang, Minghua Deng, Liqiang Nie, Zhouchen Lin, and Xian-Sheng Hua. Graph contrastive clustering. In *ICCV*, 2021.

A APPENDIX

A.1 COMPARISON BETWEEN EXISTING GNNS AND GIG

We first illustrate the main novelty of the proposed GIG compared with existing GNNs in Figure 6, where existing GNNs can only process graphs, each of whose vertices is represented by a vector/single value, while our GIG can process graphs whose vertices further contain heterogeneous graphs.

Figure 6: Comparison between GIG and existing GNNs



A.2 FORMULATIONS OF THE GIG(GAT) AND GIG(GATEDGCN)

In our experiments, the vertex and edge feature updating functions of the proposed GIG are defined according to the GAT (Veličković et al., 2017) and GatedGCN (Bresson & Laurent, 2017). Thus, we provide the detailed formulations of the GIG(GatedGCN) and GIG(GAT) as follows.

A.2.1 GIG(GATEDGCN)

For GIG models whose vertex/edge updating functions are customized from GatedGCN, its sub-edge and global-edge updating functions (i.e., the f_e^S in Eqa. 6 and f_e^G in Eqa. 8) are defined as:

$$f_e(X, Y, Z) = X + \text{LeakyReLU}(\text{BatchNorm}(W_1 X + W_2 Z + W_3 Y)) \tag{16}$$

where $W_1, W_2, W_3 \in \mathbb{R}^{D\times D}$ are learnable weight parameters. Meanwhile, both of its sub-graph level and global-graph level vertex updating functions (i.e., the f_v^S in Eqa. 7 and f_v^G in Eqa. 9) can be formulated as:

$$f_v(X, Y) = X + \text{LeakyReLU}(\text{BatchNorm}(W_4 X + Y) \tag{17}$$

where $W_4 \in \mathbb{R}^{D\times D}$ is a learnable weight matrix; and Y is the impact of adjacent vertices on the target vertex $\boldsymbol{v}_n$, which can be computed as:

$$Y = \sum_{v_m \in \mathcal{N}(v_n)} (e_{n,m} \odot W_5 v_m) \tag{18}$$

$$e_{n,m} = \frac{\exp(\hat{e}_{n,m})}{\sum_{\boldsymbol{v}_{m'} \in \mathcal{N}_{\boldsymbol{v}_n}} \exp(\hat{e}_{n,m'}) + \epsilon} \tag{19}$$

where $W_5 \in \mathbb{R}^{D\times D}$ is also a learnable weight matrix; $\odot$ is the hadamard product; and ϵ is used to avoid a zero denominator. Here, the $\boldsymbol{e}_{n,m}$ denotes the soft attention representation computed from sub-edge feature $\boldsymbol{e}^i_{n,m}$ connecting the target vertex $\boldsymbol{v}^i_n$ with its sub-vertex neighbours $\boldsymbol{v}^i_m$ within each sub-graph during the the SGU processing. During the GGU module processing, $\boldsymbol{e}_{n,m}$ denotes the soft attention representation obtained from global-edge features $\boldsymbol{e}^{i,j}_{n,m}$ connecting the target vertex $\boldsymbol{v}^i_n$ with its global-vertex neighbours $\boldsymbol{v}^j_m$. Please check Dwivedi et al. (2020); Bresson & Laurent (2017) for more detailed explanations.

A.2.2 GIG(GAT)

For GIG models whose vertex updating functions are customized from GAT, both of its sub-graph level and global-graph level vertex updating functions are defined as:

$$f_v(X, Y) = \text{Concat}^K_{k=1}(\text{ELU}(YW_6X)) \tag{20}$$

where $W_6 \in \mathbb{R}^{\frac{d}{K}\times d}$ is a learnable weight matrix; Y is defined as the attention coefficients for each head:

$$Y = \sum_{\boldsymbol{v}_m \in \mathcal{N}(\boldsymbol{v}_n)} e(k)_{n,m} \tag{21}$$

where $e(k, l)_{n,m}$ can be computed as:

$$e(k)_{n,m} = \frac{\exp(\hat{e}(k)_{n,m})}{\sum_{m' \in \mathcal{N}_{\boldsymbol{v}^i_n}} \exp(\hat{e}(k)_{n,m'})} \tag{22}$$

$$\hat{e}(k)_{n,m} = \text{LeakyReLU}(V(k)\text{Concat}(U(k)\boldsymbol{v}_n, U(k)\boldsymbol{v}_m)) \tag{23}$$

where the $V(k)$ denotes the K linear projection heads, and $e(k)_{i,j}$ denotes the attention coefficients obtained from the adjacent vertices from each head. In particular, the $e(k)_{n,m}$ denotes the attention coefficients computed from vertex neighbours within the sub-graph in the SGU module (i.e., $\boldsymbol{v}^i_m \in \mathcal{N}_{\text{sub}}(\boldsymbol{v}^i_n)$), while it denotes attention coefficients computed from vertex neighbours among sub-graphs in the GGU module (i.e., $\boldsymbol{v}^j_m \in \mathcal{N}_{\text{global}}(\boldsymbol{v}^i_n)$). Please check Dwivedi et al. (2020); Veličković et al. (2017) for more detailed explanations.

A.3 PSEUDOCODE FOR THE GIG LAYER

Algorithm 1 GIG

Input: A global graph $\mathcal{G}$ that consists of a set of sub-graphs $V_G(V_S, E_S, R_S)$ ($R_S \in V_S$) and a set of global-edges E_G, whose topology is defined by the adjacency matrix $\mathcal{A}$.

Output: A latent global-graph $\hat{\hat{\mathcal{G}}}$, consisting of a set of updated sub-graphs $\hat{V}_G(\hat{V}_S, \hat{E}_S, \hat{\hat{R}}_S)$ and a set of updated global-edges $\hat{E}_G$.

Definition: f_e^S is the sub-edge updating function, f_v^S is the sub-vertex updating function, f_e^G is the global-edge updating function; f_v^G is the global-vertex updating function; $|\cdot|$ represents the number of elements in the indicated set. $\mathcal{N}(\cdot)$ represents the neighbors of the indicated vertex.

SGU:

for $i = 1$ to $|V_S|$ **do**
  for $n = 1$ to $|V_S^i|$ and $m \neq n$ and $\mathcal{A}_{n,m}^i = 1$ to $|V_S^i|$ **do**
    $\hat{e}_{n,m}^i \leftarrow f_e^S(v_n^i, v_m^i, e_{n,m}^i)$, where $v_n^i \in V_S^i$, $v_m^i \in V_S^i$ and $e_{n,m}^i \in E_S^i$
  end for
  $\hat{E}_S^i = \{\hat{e}_{n,m}^i | \mathcal{A}_{n,m}^i = 1\}$
  for $n = 1$ to $|V_S^i|$ **do**
    $\hat{v}_n^i \leftarrow f_v^S(\boldsymbol{v}_n^i, \text{Agg}(g_v(\hat{\boldsymbol{e}}_{n,m}^i, \boldsymbol{v}_m^i) | \boldsymbol{v}_m^i \in \mathcal{N}_{\text{sub}}(\boldsymbol{v}_n^i))$
  end for
  $\hat{V}_S^i = \{\hat{\boldsymbol{v}}_n^i | n = 1, 2, \cdots, N_i\}$
end for
$\hat{E}_S = \{\hat{E}_S^1, \hat{E}_S^2, \cdots, \hat{E}_S^I\}$
$\hat{R}_S \in \hat{V}_S = \{\hat{V}_S^1, \hat{V}_S^2, \cdots, \hat{V}_S^I\}$

GGU:

while $\boldsymbol{v}_n^i \in R_S^i$ and $\boldsymbol{v}_m^j \in R_S^j$ and $\mathcal{A}_{n,m}^{i,j} = 1$ **do**
  $\hat{\boldsymbol{r}}_n^i = \hat{\boldsymbol{v}}_n^i$ and $\hat{\boldsymbol{r}}_m^j = \hat{\boldsymbol{v}}_m^j$
  $\hat{e}_{n,m}^{i,j} \leftarrow f_e^S(r_n^i, r_m^j, e_{n,m}^{i,j})$, where $r_n^i \in V_S^i$, $r_m^j \in V_S^j$ and $e_{n,m}^{i,j} \in E_G$
end while
$\hat{E}_G = \{\hat{e}_{n,m}^{i,j} | \mathcal{A}_{n,m}^{i,j} = 1\}$
while $\boldsymbol{r}_n^i \in R_S^i$ and $\mathcal{A}_{n,m}^{i,j} = 1$ **do**
  $\hat{\hat{\boldsymbol{r}}}_n^i = f_v^G(\hat{\boldsymbol{r}}_n^i, \text{Agg}(g_v(\hat{\boldsymbol{e}}_{n,m}^{i,j}, \hat{\boldsymbol{r}}_m^j) | \hat{\boldsymbol{r}}_m^j \in \mathcal{N}_{\text{global}}(\hat{\boldsymbol{r}}_n^i))$
end while
$\hat{\hat{R}}_S = \{\hat{\hat{\boldsymbol{r}}}_n^i | \boldsymbol{r}_n^i \in R_S\}$

Return: $\hat{\hat{\mathcal{G}}}(\hat{\hat{V}}_G(\hat{V}_S, \hat{E}_S, \hat{\hat{R}}_S), \hat{E}_G)$

A.4 RUNTIME ANALYSIS

We provide the runtime analysis of our GIG(GatedGCN) on seven evaluated datasets in Table 6, in terms of: (i) average running duration for each training iteration (the number of graph samples are trained in each iterntation is listed in Table 2); (ii) average running duration for each training epoch; and (iii) running duration for the model's convergence.

Table 6: Runtime analysis for experiment on seven benchmark datasets. Detailed information about two types of configurations is explained in Sec. A.7

Task	**Graph classification**		**Graph regression**		**Vertex classification**		**Edge classification**
Dataset	MNIST	CIFAR10	ZINC	AQSOL	PATTERN	CLUSTER	TSP
Type	Configuration 1		Configuration 2		Configuration 2		Configuration 1
Each iteration	0.41s	0.34s	0.68s	0.66s	1.32s	1.02s	0.45s
Each epoch	192s	160s	64.08s	51.30s	142s	585s	202s
Convergence	6.32hr	6.98hr	13.1hr	1.21hr	1.82hr	6.98hr	5.98hr

A.5 DATASET DETAILS AND STATISTICS

We provide the details of seven benchmark graph datasets that have been used in this paper in Table 7.

Table 7: Statistics of the seven employed graph datasets.

Task	Dataset	Graphs	Avg. Nodes	Avg. Edges
Graph classification	MNIST	70K	70.57	564.53
	CIFAR10	60K	117.63	941.07
Graph regression	ZINC	12K	23.16	49.83
	AQSOL	∼10K	17.57	35.76
Vertex classification	PATTERN	14K	117.47	4749.15
	CLUSTER	12K	117.20	4301.72
Edge classification	TSP	12K	275.76	6894.04

A.6 MODEL AND TRAINING DETAILS

Model settings: We first provide detailed model settings in Table 8, which represents the settings for our best GIG(GatedGCN) models whose results are reported in Table 1, where N.O.G represents the number of vertices (graph samples) used for constructing each global graph; N.O.L represents the number of stacked GIG layers. In addtion, we also provide some model internal details as: Dim(PE) represents the dimension of positional encodings; Dim(H) represents the kernel number of each hidden layer; Dim(O) represents the dimension of output layer; Dim(EO) represents the dimension of embedding layer; Dim(R) represents the dimension of readout. Please check the code of Dwivedi et al. (2020) for the definition of Dim(PE), Dim(H), Dim(EO) and Dim(R).

Table 8: Settings of our best GIG models for seven benchmark datasets.

Dataset	N.O.G	N.O.L	Dropout	Dim(PE)	Dim(H)	Dim(O)	Dim(EO)	Dim(R)
MNIST	150	1	0.15	-	85	85	-	3
CIFAR10	128	1	0.12	-	85	85	-	2
CLUSTER	21	3	0.03	10-20	85	85	9	3
PATTERN	128	2	0.00	2	85	85	9	3
ZINC	128	1	0.00	4	85	85	9	3
AQSOL	128	1	0.00	4	85	85	9	3
TSP	27	3	0.01	-	85	85	-	3

Training details: We then provide the training details for achieving our best models on seven datasets in Table 9, where all models are trained with the AdamW (Loshchilov & Hutter, 2017) optimizer. In addition, we provide the training and validation loss curves of our best models in Figure 7 and 8.

Table 9: Training details for 7 benchmark datasets.

Dataset	Learning rate	Loss	Decay strategy	Optimizer
MNIST	0.0025	Cross entropy	CosineAnnealingWarmRestarts	AdamW
CIFAR10	0.0023	Cross entropy	CosineAnnealingWarmRestarts	AdamW
CLUSTER	0.0036	Cross entropy	CosineAnnealingLR+ConstantLR	AdamW
PATTERN	0.0030	Cross entropy	CosineAnnealingLR	AdamW
ZINC	0.0027	L1	CosineAnnealingWarmRestarts	AdamW
AQSOL	0.0023	L1	CosineAnnealingWarmRestarts	AdamW
TSP	0.0036	Cross entropy	ReduceLROnPlateau	AdamW

Figure 7: Training and validation losses achieved for MNIST, CIFAR10, CLUSTER and PATTERN datasets.

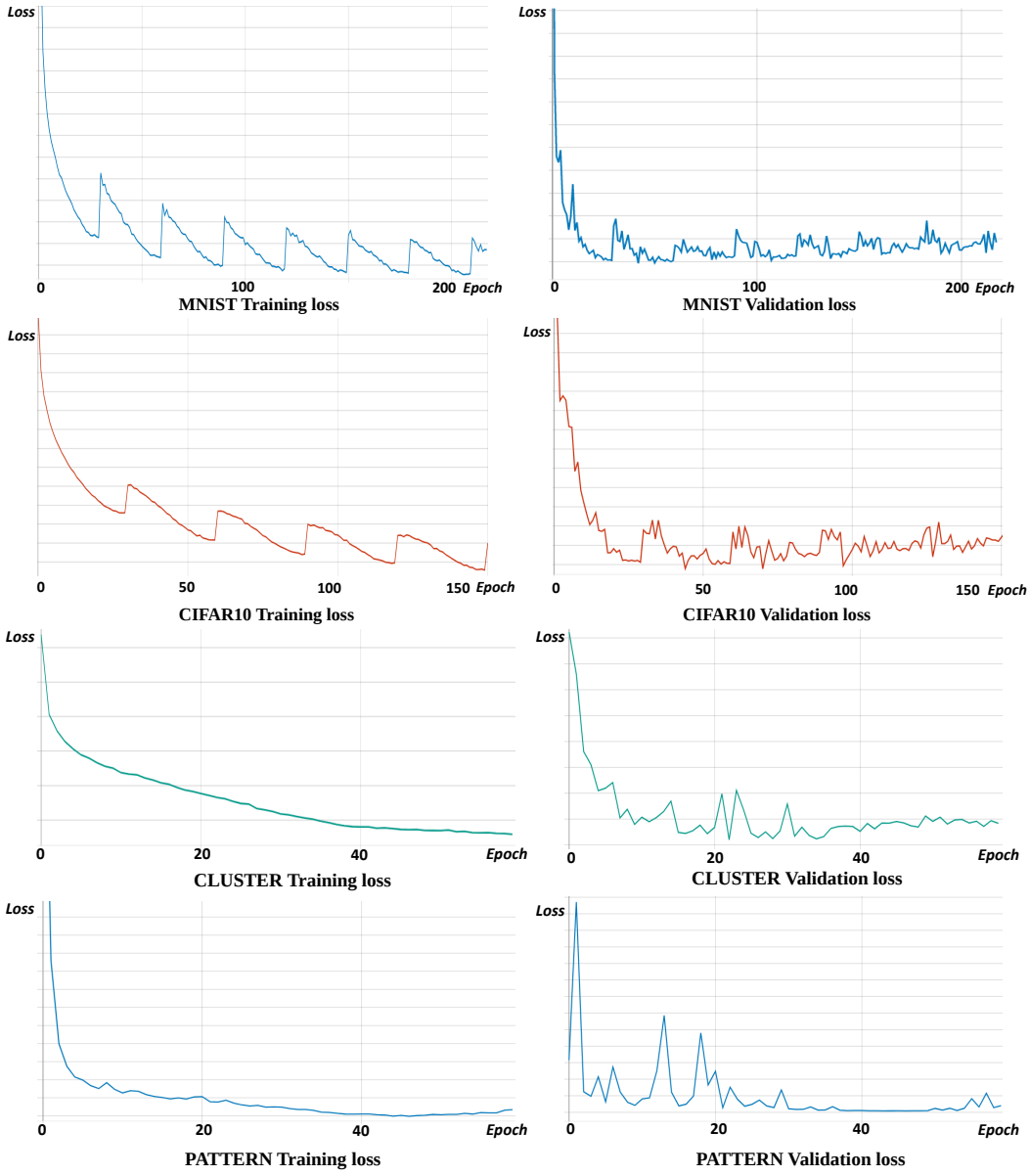
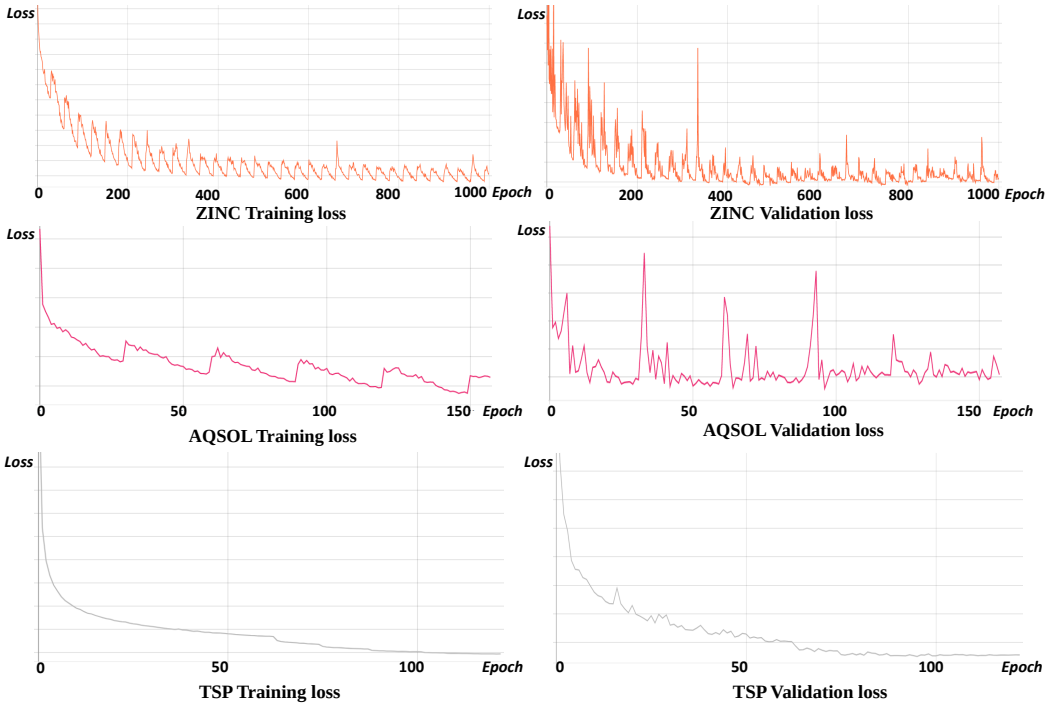


Figure 8: Training and validation losses achieved for ZINC, AQSOL, and TSP datasets.



A.7 SOFTWARE AND HARDWARE SETTINGS

All experiments were implemented using the Deep Graph Library (DGL) (Wang et al., 2019a) and PyTorch (Paszke et al., 2019) numerical library. Two sets of hardware configurations were used for training. The first set contained 1 NVIDIA A100 GPU with 80GB RAM/GPU, 2 15-core Intel(R) Xeon(R) Platinum 8358P CPUs @ 2.60GHz and 240GB RAM, which was utilized to train MNIST, CIFAR10 and TSP datasets, and the second set contained 1 NVIDIA A100 GPU with 80GB RAM/GPU, 1 16-core Intel(R) Xeon(R) Platinum 8369B CPU @ 2.90GHz and 120GB RAM, which was utilized to train CLUSTER, PATTERN, ZINC and AQSOL datasets.